Sadip Giri

Closure

I was attracted to learn my first ever perplexing and powerful language called Clojure; Lisp on the JVM. I noticed that Clojure was the richest language. Here, “richest” means like Scala - Clojure runs in a commercial deployment platform: JVM, as Clojure is functional: I was able to apply advanced concepts to my code, and most importantly it’d be expressed in terms of Lisp. I was fascinated to witness my Clojure’s code being concise, easier to read, and more fun to write in a completely different way as Clojure was dynamically typed. Moreover, I was intrigued to see safe and concurrent access to memory as Clojure transactional memory worked like transactional databases. I noticed an interesting data type called Ratio in Clojure which allowed me to delay computation to avoid the loss of precision. I think this is crucial. Clojure’s basic building block called Form was really effective as it helped me to break the program down into pieces such as booleans, characters, strings, sets, maps, and vectors. The nicest thing about Clojure was that I was able to access any portion of any argument as a parameter because the Binding was super easy in it. In addition to that, the use of destruction allowed me to bind any parameter to any part of an inbound argument. Although Clojure contained a lot of abstractions in the data structures and library, I really liked Clojure emphasizing language-on-platform rather than language-is-platform. I was dazzled to learn the concept of Lazy Evaluation (lazy sequences) which allows us to compute an

infinite sequence (just like we do in Math). In the same way, by using Defrecord and Protocols - I was able to implement types that were full citizens on the JVM. And I was conspired to master the new concept called Macro Expansion which helps us to redefine the language through powerful combination. I found it interesting to see how Clojure maintained consistency and integrity using STM (Software Transactional Memory). Furthermore, I discovered simple and safe ways to synchronously handle mutable state. That is, through the use of Atoms and Refs.

It was very difficult to understand Clojure without understanding Lisp as most of the programmer says it has same limitations as well as same considerable strengths as that of Lisp. Although Clojure was a powerful language to learn, I found it weird to enclose any function call entirely in parentheses. It was awkward to see Math being expressed in Prefix notation as I preferred, and was used to Infix notation. I was also confused to see lists being used for code and vectors for data. Due to limitations of JVM, Clojure does not support implicit tail recursion optimization; which I observed as one of the main disadvantages of it because it leads to continuous consumption of memory until all the memory is exhausted. Though Clojure contained a lot of advanced and powerful elements, I found it sophisticated because it took me a lot of time to explore and understand those concepts. I also saw it demanding as a developer because it contains so many abstraction tools and concepts which were overwhelming. That is, the learning curve was oppressive.

I strongly believe that Clojure is the most powerful and rich language that I have ever encountered. So, I'd definitely come back to use it if I wanna build better and faster software in Clojure.