Clojure for JavaScript Developers

A Short Introduction

Clojure is a Lisp-based functional programming language created in 2007 by Rich Hickey.

Clojure for the Brave and True boldly states “become a better programmer” as its subtitle, and while that’s quite a grand statement, I agree that as a developer, you will grow if you learn functional programming. I also think that while JavaScript supports a lot of functional aspects, it’s a cumbersome language in which to learn and use

and use functional paradigms.

If you're not familiar with functional programming, the rest of this article may be difficult to read. But functional programming is important, and teaches you new ways to think. So take the time to learn the new paradigm and a new programming language, even if it isn't Clojure.

The syntax may seem very foreign, but don't let that scare you off. Sit with it and become comfortable using it, and you'll be rewarded with a rich, elegant, and concise programming language.

Clojure vs ES6

If you're going to use ES6 in production, you're going to be using a transpiler to convert your code to JavaScript that is compatible with older browsers. If you're going to use a transpiler anyway, why not use a language that is richer, more concise, and more elegant than JavaScript?

I have written JS projects in both functional and imperative styles, and my preference is strongly for

Clojure and ClojureScript

Clojure is built to run on any Java Virtual Machine (JVM). ClojureScript is a dialect of Clojure that can be compiled down to JavaScript. The differences between Clojure and ClojureScript are very minor.

It's even possible to invoke functions in other JavaScript libraries from within ClojureScript. A trivial example:

```
window.helloWorld = function() {
  console.log("Hello, world!");
}
```

```
(.helloWorld js/window);
```

For more, read about ClojureScript: JavaScript Interop.

Paradigms

Immutability: A core concept of Clojure is that of immutability - once you create some state (such as a list), you don't make changes to it. Functions that act on that state dervice a new state from the given one. This helps preserve function purity. Clojure goes one step even further and separates *identity* from *state*. This is best explained by the article on th clojure website: http://clojure.org/about/state

Thread-safe/Concurrency: Another core concept of Clojure is thread-safe, concurrent programming. Using what they've dubbed the *Software Transactional Memory*

an article here: http://clojure.org/about/concurrent_programming

Sequences: Clojure uses something similar to a standard iterator, known as a sequence, which provides a well-defined interface to traverse a list. Unlike iterators, which contain state that points into a list, sequences are immutable and persistent, and they are also consumed lazily. Read more: http://clojure.org/reference/sequences

Clojure Features

Sets

Clojure supports sets as a built-in datatype. Sets are guaranteed to hold only unique values. These are also present in JavaScript ES6.

```
(set [1 1 2 2])
; => #{1 2}
```

Map

The map function is a built-in special-form, whereas in JavaScript ES6 it is only available on the `Array` type.

```
(map triple [1 2 3])
; => (3 6 9)
```

JavaScript ES6 it is also available on the `Array` type.

```
(reduce + [1 2 3])
; => 6
```

Function docs

When you define a function, you can provide the documentation right as a parameter in the definition. It can then later be referenced using `(doc function-name)`. This makes it easy to look up documentation when you are experimenting in the REPL.

```
(defn triple
  "triples the given value"
  [value]
  (* 3 value))
```

Arity overloading

You can provide multiple different signatures for the same function, and the one that matches the arguments when invoked will be called.

```
(defn area
  "calculate area. with one parameter, it assumes a square"
  ([x]
    (* x x))
  ([x y]
    (* x y)))
(area 2 3)
; => 6
```

[& args] / Variable arguments (aka variadic functions)

Clojure also provides support for variadic functions, or functions that take an arbitrary amount of arguments. JavaScript supports this as well, though in a rather ugly way, through the `arguments` object.

```
(defn count-arguments
  "Returns the number of arguments passed"
  [ & args ]
  (count args))
```

Argument destructuring ([[first]])

When declaring function arguments, you can declare a sequence structure, and the values will be properly populated. In this example, the first three values of the given list will be assigned to `first`, `second`, and `third`.

```
(defn third
  "return the third argument of the given collection"
  [[first second third]]
  third)
(third '(1 2 3 4))
; => 3
```

Learning More

This short introduction is just the tip of the iceberg. Clojure is being used in production by several prominent companies, including Atlassian, ThoughtWorks, and Puppet. It's a rich programming language with large

Clojure.org Getting Started Guide A shorter introduction than Clojure for the Brave and True, but it quickly delves deeply into some of Clojure's more unique features.

Clojure for the Brave and True Learn Clojure by smacking hobbits. An excellent, whimsical introductory book that delves deeply into Clojure. A great reference, though I can't condone violence against hobbits. Note that they use primarily JavaScript for some of their examples.

Om, a ClojureScript interface to React.js If you want to use ClojureScript for modern web development, Om provides an interface into React.

Team / Join us

GitHub

298 Dundas St. West, Unit B, 2nd Floor

Toronto, Ontario

M5T 1G2, Canada