Enhancing Your Libraries' Security and Performance through Macros

---

Veit Heller
Clojure Meetup Berlin

November 13, 2019

TL;DR

"Just remember that macro programming is not about style; it is about power."
— Doug Hoyte, Let Over Lambda (Chapter 3)

TL;DR, but really

Power has to be yielded or wielded in favor of others, not yourself.

Why not macros?

Macros cause problems as often as they solve them.
They can be impenetrable, infectuous, and stylistically oppressive.

Really?

You're framing it wrong.

Don't use macros to extend your syntactic and expressive power.

First, use them to make it harder to make mistakes.

```
(defmacro fmt [s & args]
  (let [a (dec (count (filter #(not= "" %)
                              (str/split s #"%"))))
        b (count args)]
    (if (= a b)
      `(apply format ~s '~args)
      (exc "arguments to format don't match (expected " a
           ", got " b ")"))))
```

Listing 1: A safer format

Avoid read-eval, and catch things statically if you can.

Then, use them to say what you mean.

```
const float sine_lut[LUT_SIN_SIZE + 1]={
  1, 0.999981175282601, 0.999924701839146, 0.999830581795823,
  0.999698818696204, 0.999529417501093, 0.99932238458835,
  0.999077727752645, 0.998795456205172, 0.998475580573295,
  // 1013 more constants[...]
  1.0
};
```

Listing 2: A sine lookup table.

```
// build with:
// for(i=0; i<LUT_SIN_SIZE; i++){
//   sine_lut[i] = sin(2PI * i / LUT_SIN_SIZE);
// } sine_lut[LUT_SIN_SIZE] = 0.0;
```

Listing 3: A sine lookup table, builder.

```
(defmacro build-sine [sym size &{ :keys [start]
                                  :or {start 0}}]
  `(def ~sym
    ~(vec (map #(Math/sin (/ (* % (* 2 Math/PI)) size))
               (range start (+ size start))))))

(build-sine *sine-lut* 1024)
```

Listing 4: A sine lookup table, computed.

Before you optimize, measure. Not just time, but memory as well!

And lastly, for the sake of yourself and others, use them to make better macros.

```
(defmacro defmacro! [name args & body]
  (let [rm (into {}
                 (map (fn [s] [s `(quote ~(gensym))])
                      args))]
    `(defmacro ~name ~args
       `(let ~~(into [] (mapcat reverse rm))
          ~(clojure.walk/postwalk-replace ~rm ~@body)))))
```

Listing 5: A macro utility—or, as Alan Perlin said: "Syntactic sugar causes cancer of the semicolon."

Macro-writing macros and nested backquotes are extremely powerful, but hard to master. And again, think of your error messages!

References

- Greg Hendershott: Fear of Macros
- Doug Hoyte: Let Over Lambda (security in chapter four)
- Colin Jones: Mastering Clojure Macros (performance in chapter four)
- Paul Graham: On Lisp (but also read the Graham Crackers)
- These slides: `https://github.com/hellerve/talks`
- A series of blog posts on Scheme macros: `https://blog.veitheller.de/scheme-macros`

Thank you!

Questions?

Slides at `https://github.com/hellerve/talks`