U46LFMYTD : my code takes 7 days to run ^^
U0LGCREMU : haha! didn't i tell you? :)
U5ZAJ15P0 : <@U0LGCREMU> you did! that's how I vaguely remembered there was a function for this, but I have the brain of a goldfish today so I couldn't think of the name
U09LZR36F : Is there a form of resolved keywords which looks into `refer`'d vars?
```
(ns foo
  (:require [clojure.string :refer [blank?]]))

(println ::blank?)
;; (desired) =&gt; clojure.string/blank?
;; (actual)  =&gt; foo/blank?
```

U1B0DFD25 : <@U5ZAJ15P0> there's also `sayid`: <https://github.com/bpiel/sayid>
U060FKQPN : keywords have no relationship with vars <@U09LZR36F>
U1B0DFD25 : <@U09LZR36F> is it something you need for spec?
U0567Q30W : <@U050R7ECY> Cursive has a bytecode debugger, and you can definitely see the locals (as well as other normally invisible locals created by e.g. destructuring). It also does its best to help control locals clearing.
U0567Q30W : You're right AFAIK that CIDER's debugger is source transformation.
U09LZR36F : I figured it would be useful for that, yes
U628K7XGQ : Going back to my question from this morning to introspect a Java object, here's a fn I've written to expose the fields of an object via reflection (clojure.reflect really didn't help).I've crammed it all into a single fn and it looks quite ugly. Any suggestions on how to improve the code style. I'm still a clnewb?

```
(defn java-&gt;map
  "Turns fields of a Java object into a map, up to 'level' deep"
  ([obj] (java-&gt;map obj 1))
  ([obj level]
   (when (some? obj)
     (let [c (class obj)]
       (cond
         ;;;(.isPrimitive c) obj never works because clojure implicitly wraps primitives
         (contains?
           #{java.lang.Long java.lang.Character java.lang.Byte
             java.lang.Double java.lang.Float java.lang.Short java.lang.Integer} c) obj
         (= 0 level) (.toString obj)
         (instance? java.lang.String obj) obj
         (.isArray c) (concat
                       (-&gt;&gt; obj
                            (take 5)
                            (map (fn [e] (java-&gt;map e (dec level)))))
                       (when (&gt; (count obj) 5) [:more (count obj)]))
         :else
         (assoc (into {} (-&gt;&gt; (concat (.getDeclaredFields c)
                                       (.getFields c))
                                  (filter #(= (bit-and (.getModifiers %) java.lang.reflect.Modifier/STATIC) 0)) ;;; ignore static fields
                                  (map
                                    #(do (.setAccessible % true)
                                         [(keyword (.getName %))
                                          (java-&gt;map (.get % obj) (dec level))])))))
           :-type c  ;;; add the type as well
           ))))))
```

Examples:
```
(java-&gt;map (java.util.Date.) 4)
=&gt; {:fastTime 1500771066642, :cdate nil, :-type java.util.Date}
```

(java->map (java.text.AttributedString. "bubu") 2)
=>
{:text "bubu",
 :runArraySize 0,
 :runCount 0,
 :runStarts nil,
 :runAttributes nil,
 :runAttributeValues nil,
 :-type java.text.AttributedString}
```

U2TCUSM2R : I'm having trouble adding metadata inside a macro. It's a bit weird (as usual) since it overrides `defn` in order to capture the ast after compilation, but works otherwise: ```(defmacro defn [name &amp;amp; decls] `(def ^{:ast ~decls} ~name (fn ~decls)))```
U051SS2EU : defmacro never sees the reader macro, it's applied before it sees the form
U051SS2EU : you need to use with-meta or vary-meta to add metadata to the symbol you emit
U2TCUSM2R : I did try that, but perhaps didn't do it correctly
U2TCUSM2R : (and good call because I definitely don't want this done at runtime)
U051SS2EU : `^{:foo :bar} baz ` translates to `(with-meta baz {:foo :bar})`
U2TCUSM2R : right
U2TCUSM2R : lemme give it another go and post the results if it doesn't work
U2TCUSM2R : ok, so yeah so for some reason the actual `fn` part doesn't compile when i write it like that: ```(defmacro defn [name &amp;amp; decls] `(def ~(with-meta ~name {:ast ~decls}) (fn ~decls)))```
U051SS2EU : nested ~ isn't valid like that
```