U628K7XGQ : How are you guys printing the guts of Java objects on the REPL? I'd love to be able to convert any java object to a graph of maps I can easily inspect.(similar to what the debugger in IntelliJ allows you to do)

U060FKQPN : `bean`
U628K7XGQ : wouldn't work on non-beans
U628K7XGQ : I tried to make sense of `clojure.reflect` but before I spent my time on that - shall we say - sparsely documented lib, I was wondering if someone had a neat `java-to-map` fn
U050MP39D : I think clojure.reflect/reflect basically does that iirc
U051SS2EU : if I recall, bean used to be more general and didn't look for JavaBean specific stuff... let me see if I can find it
U051SS2EU : ```+user=> (bean (java.util.Date.)){:day 6, :date 22, :time 1500741217657, :month 6, :seconds 37, :year 117, :class java.util.Date, :timezoneOffset 420, :hours 9, :minutes 33}
```

U051SS2EU : maybe it's that Date is a bean and I never realized it was?
U050MP39D : yeah bean looks quite good```
(bean (HTableDescriptor. (TableName/valueOf "foo")))
=&gt;
{:familiesKeys #{},
 :columnFamilies #object["[Lorg.apache.hadoop.hbase.HColumnDescriptor;"
                         0x124eda1b
                         "[Lorg.apache.hadoop.hbase.HColumnDescriptor;@124eda1b"],
 :regionReplication 1,
 :memStoreFlushSize -1,
 :name #object["[B" 0x19dca84c "[B@19dca84c"],
 :tableName #object[org.apache.hadoop.hbase.TableName 0x6328ce6a "foo"],
 :metaRegion false,
 :rootRegion false,
 :compactionEnabled true,
 :maxFileSi... snip
```

U051SS2EU : and you could start with bean's source code if you want something that isn't so JavaBean centric
U065JNAN8 : I don't understand this line in the docs about clojure.spec
"Thus a bare (s/keys) is valid and will check all attributes of a map without checking which keys are required or optional."

What is it checking if there are no specs to check?

```
(s/def ::foo (s/keys))
(s/valid? ::foo {:yolo 42})
=&gt;true
```

U065JNAN8 : Ah I hadn't read enough. Namespace qualified keys will be checked against their specs
U050R7ECY : Are there any good debuggers that work at the bytecode level? I have cider, but I have an infinite loop somewhere, so I need 'pause execution'
U051SS2EU : cider and cursive have this - to some degree, you need to turn off some clojure features for it to work well
U051SS2EU : usually I opt for adding (swap! debug-atom conj {:context ::foo :data foo}) in the middle of some function and then use the resulting atom in the repl to figure out what's going on (which does require iterating sometimes to figure out which data you should even be tracking of course)
U5ZAJ15P0 : <@U051SS2EU> is there an equivalent of Ruby's "pry" in clojure? e.g. a way to open a repl at any point in a program, with access to local bindings?
U050R7ECY : I thought cider's debugger worked by instrumenting and evaling source?
U051SS2EU : <@U050R7ECY> I didn't check recently but at one point they were actually using jdx bindings to get into the byte code level - when I saw people claim "cider can debug like cursive now" I assumed that meant that feature was working
U051SS2EU : cursive definitely does byte code level debugging
U051SS2EU : <@U5ZAJ15P0> that's hard with a compiled language

U051SS2EU : I don't think that exists, definitely not in a general way
U050R7ECY : <@U051SS2EU> that doesn't look to be in the latest release: <https://cider.readthedocs.io/en/latest/debugging/>
U051SS2EU : <@U050R7ECY> oh, clearly I was misinformed and/or misunderstood what I was being told
U051SS2EU : thanks for the clarification
U050R7ECY : <@U5ZAJ15P0> cider can do that by recompiling functions. Macros get access to all local variables in scope when compiling
U050R7ECY : i.e. `(let [x 1] (my-dbg (inc x))`, if my-dbg is a defmacro, there's an extra `&amp;env` variable that contains `{x 1}`
U5ZAJ15P0 : Well, I guess I should start using emacs then
U051SS2EU : I find it easier to just use a macro that puts the locals in a hash map, then access it <https://gist.github.com/noisesmith/3490f2d3ed98e294e033b002bc2de178>
U050R7ECY : a bytecode debugger can see the locals in scope, but IIRC the compiler throws away the variable names in the bytecode, so you just get `foo_1`, etc.
U050R7ECY : and there's locals clearing
U051SS2EU : <@U050R7ECY> right that's the clojure feature I was t alking about - you can disable it
U46LFMYTD : Has anyone used visualvm to profile clojure code?
U46LFMYTD : I cant get it to work
U051SS2EU : yes - it works but yourkit is much better
U050R7ECY : try the sampler rather than the profiler
U050R7ECY : profiler rarely works well for me
U060FKQPN : it doesn't
U060FKQPN : the compiler, I mean
U051SS2EU : but both are somewhat difficult because so much of profiling assumes that the class you are looking at is important, and most of the classes are going to be eg. clojure.lang.PeristentVector
U060FKQPN : the local names are right there in the bytecode
U46LFMYTD : hmm
U050R7ECY : sure, but at least the sampler comes back with *a* result :slightly_smiling_face:
U051SS2EU : it's definitely doable though - I've had a lot of help from profiling
U050R7ECY : profiler usually takes forever to instrument and then doesn't work
U46LFMYTD : well, I've been following the guide written here: <https://torsten.io/stdout/how-to-profile-clojure-code/>
U051SS2EU : <@U050R7ECY> oh, I haven't had that issue
U051SS2EU : it needs more time to start up for sure though
U46LFMYTD : but it says Failed to Create JMX connection to target application
U051SS2EU : then you need to add the args to java to allow jmx connect
U5ZAJ15P0 : is there any good way to debug clojure without using a particular editor? (aka no cursive and no cider)
U051SS2EU : that's where swapping data into an atom excels
U051SS2EU : or you could use jdb - if you are familiar enough with clojure internals it can work
U051SS2EU : but it's confusing
U5ZAJ15P0 : <@U051SS2EU> funnily enough swapping data into an atom is the solution I came up with this morning
U051SS2EU : there's an old tool for setting up a "debug repl" that kind of worked but it was funky
U050R7ECY : <@U46LFMYTD> are you specifically trying to get remote profiling working? Local clojure process should just work
U051SS2EU : <@U5ZAJ15P0> as a level up from swapping into an atom, I also dump the relevant part so I can use it in a unit test when I figure out what is going on - I made a small lib for this <https://github.com/noisesmith/poirot>
U5ZAJ15P0 : <@U051SS2EU> oh yes, you sent that one to me already but hadn't yet looked into it
U051SS2EU : <@U46LFMYTD> oh - weird - I haven't experienced that issue, not sure what that's about
U46LFMYTD : :disappointed: Ive had this working on a remote process before... something about running it locally is not working
U050R7ECY : ok, got `jdb` working acceptably well
U46LFMYTD : is there any special setup in the project.clj to use YourKit? It seems a lot of feature are missing when i connect to my running repl
U051SS2EU : you might need to add jvm args to allow the kind of connection it wants
U051SS2EU : those would be the same args a java program would need
U051SS2EU : I've had a lot of luck with using the yourkit agent
U051SS2EU : <@U050R7ECY> I just realized this would be relevant - if you send a signal you can make the jvm print all its stack traces (or you can use jstack from another terminal to get all stack traces by pid) and if you have a loop that isn't exiting a few stack traces should show where that's happening pretty quickly
U050R7ECY : yeah, I did basically the same thing using jdb
U051SS2EU : I forget the signal name but on *nix terminals its bound to `C-\`

U050R7ECY : using `threads` and `where`
U051SS2EU : cool
U5ZAJ15P0 : Can someone please remind me of the name of the clojure function which does the following: `(f g :a :b :c) => [(g :a) (g :b) (g :c)]` ?
U5ZAJ15P0 : ah I am stupid, I could just map
U050R7ECY : looks almost like `juxt`
U5ZAJ15P0 : but `juxt` is what I was looking for
U5ZAJ15P0 : thanks!
U5ZAJ15P0 : my example was just wrong
U5ZAJ15P0 : thanks <@U050R7ECY>
U46LFMYTD : seems to be working much better than visualVM
U46LFMYTD : its hard to make sense of though
U46LFMYTD : in the sense that it is hard to find which of my functions is taking the longest time amid all the clojure / java functions
U051SS2EU : yeah - it takes a while to figure out, I have a long term plan to do a talk on this topic
U46LFMYTD : sounds good
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-&gt;map (java.text.AttributedString. "bubu") 2)
=&gt;
{: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 & decls] `(def ^{:ast ~decls} ~name (fn ~decls)))```
U051SS2EU : defmacro never sees the reader macro, it's applied before it sees the form