U0NCTKEV8: (are valid keywords only those keywords that can be produced by the reader?)

U0NCTKEV8 : but generally, turning keys in to keywords all the time is something every project (I have worked on)

does, and I hate it

U0NCTKEV8: but if you do it you are doing the same thing that everyone does

 ${\tt U050MP39D:"} valid keywords" are those that follow the rules in the docs imo, but the reader will create some keywords$

that don't if asked

U0NCTKEV8: which docs?

U050MP39D: https://clojure.org/reference/reader#_literals>

U0NCTKEV8 : those a docs on keywords that the reader can read

U0NCTKEV8: (which is why they are under the reader section)

U0NCTKEV8: I've seen this discussion play out over and over again in chat, on the mailing list, and in jira issues, and it

seems like the core team's general stance is what the reader excepts doesn't define what is valid

U050MP39D: the reader accepts more than those docs, but those docs do give a guarantee of what the reader will accept. so for me they make sense as a definition of valid. not the same thing as "whatever the reader accepts"

U050MP39D : eg``` cljs.user=> :&

:&

U050MP39D : but it's not documented that that will always work (or become reserved for some special system/core usage) so I wouldn't use it

U0NCTKEV8: https://dev.clojure.org/jira/browse/CLJ-1527

U0NCTKEV8: the point is, just because the reader only constructs certain kinds of symbols and keywords, that doesn't restrict the Symbol and Keyword data types to only construct those types of symbols and keywords

U0NCTKEV8: https://dev.clojure.org/jira/browse/CLJ-17

U0NCTKEV8: https://dev.clojure.org/jira/browse/CLJ-2009

U0NCTKEV8: etc etc etc

U04V70XH6: And at one point, very specifically, the Clojure/core team made `:1` (numeric keywords) illegal in the reader on the grounds that they weren't sanctioned by the docs -- and they broke a bunch of code out there in the wild so they reverted the reader change.

U04V70XH6: I was affected by that change (one of the OSS libraries I maintain) but I would have been perfectly happy to change `:1` to `(keyword "1")` -- which they said they would _not_ break... but they still reverted the change :slightly smiling face:

U04V70XH6: So we have prior art for the reader being publicly known to accept things that are not considered "correct" and not getting fixed -- and a statement that `keyword` (and, similarly, `symbol`) being able to produce "junk" output from arbitrary input: neutral face:

U0NCTKEV8: the issue isn't that the reader and the reader docs are not in 100% agreement about what is a readable keyword (that is another issue) the issue is that a valid readable keyword is not the same as a valid keyword

U0NCTKEV8: for example, the keywords json libraries create for map keys never pass through the reader

U0NCTKEV8: so they can be any crazy thing

U04V70XH6: Exactly my point, yes.

U0NCTKEV8 : which negates the purpose of turning them in to keywords in the first place, because the only reason people turn them in to keywords is since most maps in clojure use keyword keys, they more comfortable with them

U04V70XH6: Speed of lookup over plain strings. In theory.

U0NCTKEV8 : maybe

U04V70XH6: `(identical?:abc (keyword (str "a" "bc")))` => true

U04V70XH6: So map lookup will be faster if it can check for `identical?` before falling back to "egual".

U0NCTKEV8: but do you want to be interning json coming in over the wire?

U3L6TFEJF: what am I missing here?"

user=> (defn foo [args] (apply hash-map args))

#'user/foo

user=> (binding [*data-readers* {'foo #'user/foo}] #foo [1 2 3 4])

java.lang.RuntimeException: No reader function for tag foo

clojure.lang.LispReader\$ReaderException: java.lang.RuntimeException: No reader function for tag foo

java.lang.RuntimeException: Unmatched delimiter:)

clojure.lang.LispReader\$ReaderException: java.lang.RuntimeException: Unmatched delimiter:)

•

U04V70XH6: Good question, given that converting arbitrary strings to keywords can open you up to a denial of service attack.

U0NCTKEV8: I mean, you will likely be fine, and keyword's intern a weakreference, so the they will get cleared

U04V70XH6: Ya, but...:slightly_smiling_face:

U051SS2EU: ```user=> (defn foo [args] (apply hash-map args))#'user/foo user=> (binding [*data-readers* {'foo #'user/foo}] (read-string "#foo [1 2 3 4]")) {1 2, 3 4}```

U3L6TFEJF: <@U0NCTKEV8> ahh, so you need this? `(binding [*data-readers* {'foo #'user/foo}] (read-string "#foo [1 2 3 4]"))`

U3L6TFEJF: haha, timing: smile:

U0NCTKEV8: yes

U3L6TFEJF: while on the topic of tagged literals: is there anything special you need to do to get it to work with `tools.namespace`?