# Calling java from Clojure

leave a comment »

3 Votes

### Importing Java Class

In repl, when you want to import one Java class you can do

```
(import 'java.util.Date)
```

When you want to import more Java classes from a same package you can do

```
(import [java.util Date HashMap])
```

```
(ns com.techbehindtech.java
        (:import [java.util Date HashMap]))
```

### Creating Instances

```
(import 'java.util.Date)

(def today (new Date))

;; or

(def today (Date.))
```

### Calling Java instance methods

```
user> (import 'java.util.Date)
java.util.Date
user> (let [today (Date.)]
        (.getTime today))
1286749020847
```

### Calling Java static methods

```
user> (System/currentTimeMillis)
1286847946813
```

### Sugar sytax:

***Doto:***

You want to write a function that will return UTC Java Calendar object set at a specific time.

```
user> (import [java.util Calendar TimeZone Date])
user> (defn utc-time [d]
        (let [cal (Calendar/getInstance)]
              (.setTimeZone cal (TimeZone/getTimeZone "UTC"))
              (.setTime cal d)
          cal))

#'user/utc-time
```

The let block is ugly. We could use doto to make this code better.

```
user> (import [java.util Calendar TimeZone Date])
user> (defn utc-time [d]
          (doto (Calendar/getInstance)
              (.setTimeZone (TimeZone/getTimeZone "UTC"))
              (.setTime d)))

#'user/utc-time

```

***Dot Dot:***

Sometimes in java you want to make calls in chain

*Bad way*

```
user> (.length (.getProperty (System/getProperties) "user.country"))
2

```

*Better way*

```
user> (. (. (System/getProperties) getProperty "user.country") length)
2
```



**Tags**

agile build clojure clojurescript code commons logging compojure crawler data structure dependency development distributed example functional future GELF git google closure graylog2 guidelines interopability java javascript jquery json jvm kanban lean Leiningen libraries log4j logback logging logstash macros maven process protocols record REST ring sandbar scalability slf4j snippet software structure test-is testing tools web web development xml

**Recent Posts**

JSON logging using Logback
Getting Started with ClojureScript
Logging in Clojure / JVM – Part 4
Logging in Clojure / JVM – Part 3
Logging in Clojure / JVM – Part 2
Logging in Clojure / JVM – Part 1
Lean Software Development
Clojure MindMap
Introduction to Clojure Web Development using Ring, Compojure and Sandbar
Leiningen – Adding git submodule to Leiningen project

**Top Posts**

JSON logging using Logback
Calling java from Clojure
Lean Software Development
Map, Reduce and Filter in Clojure
Introduction to Clojure Web Development using Ring, Compojure and Sandbar
Logging in Clojure / JVM – Part 1
Logging in Clojure / JVM – Part 2
Compojure Demystified with an example - Part 6
Clojure Macros Simplified
Logging in Clojure / JVM – Part 3

**My tweets**

RT @clojure_conj: Clojure/conj thanks Platinum sponsor Staples Innovation Lab! staplesslabs.com/innovation-lab... #clojure_conj 1 month ago

*Even Better*

```
user> (..
       (System/getProperties)
       (getProperty "user.country")
       (length))
2
```

### Avoiding Reflection

By default jvm will be using reflection to identify type. Reflection is slow. But we can give type hints that way jvm does not have to use reflection. For example we can rewrite utc-time with type-hints like this

```
user> (set! *warn-on-reflection* true)
true
user> (defn str-length [s] (.length s))
Reflection warning, NO_SOURCE_FILE:1 - reference to field length can't be resolved.
#'user/str-length
user> (defn str-length [#^String s] (.length s))
#'user/str-length

```

### Implementing interfaces and extending classes

Let us implement Java Runnable interface

```
user> (proxy [Runnable] []
        (run []
             (println "running ...")))
#<Object$Runnable$36fc6471 user.proxy$java.lang.Object$Runnable$36fc6471@6dd33544>
```

In clojure 1.2, you could use reify macro to implement. In fact it is better than using proxy.

```
user> (reify Runnable
             (run [this]
                  (println "running ...")))
#<user$eval1664$reify__1665 user$eval1664$reify__1665@574f7121>
```



**Related**

Logging in Clojure / JVM – Part 1 — In "Clojure"

Logging in Clojure / JVM – Part 2 — In "Clojure"

Logging in Clojure / JVM – Part 4 — In "Clojure"

Written by Siva Jagadeesan
October 11, 2010 at 10:43 pm

Posted in Clojure
Tagged with clojure, interopability, java

« Clojure Macros Simplified

Testing Clojure Code – Awesome "are" »

## Leave a Reply

Enter your comment here...