In Clojure 1.3, How to read and write a file

I'd like to know the "recommended" way of reading and writing a file in clojure 1.3 .

1. How to read the whole file
2. How to read a file line by line
3. How to write a new file
4. How to add a line to an existing file

file clojure io

edited Dec 28 '15 at 0:24
Chris Martin
18.5k 3 35 71

asked Oct 13 '11 at 15:54
jolly-san
947 2 8 9

protected by Kermit Jul 5 '14 at 0:35

This question is protected to prevent "thanks!", "me too!", or spam answers by new users. To answer it, you must have earned at least 10 reputation on this site (the association bonus does not count).

1 First result from google: lethain.com/reading-file-in-clojure – jcubic Oct 13 '11 at 16:14

6 This result is from 2009, some things have been changed lately. – Sergey Oct 25 '11 at 21:00

8 Indeed. This StackOverflow question is now the first result on Google. – mydoghasworms Apr 18 '13 at 7:11

4 Answers

Assuming we're only doing text files here and not some crazy binary stuff.

Number 1: how to read an entire file into memory.

```
(slurp "/tmp/test.txt")
```

Not recommended when it is a really big file.

Number 2: how to read a file line by line.

```
(use 'clojure.java.io)
(with-open [rdr (reader "/tmp/test.txt")]
  (doseq [line (line-seq rdr)]
    (println line)))
```

The `with-open` macro takes care that the reader is closed at the end of the body. The reader function coerces a string (it can also do a URL, etc) into a `BufferedReader`. `line-seq` delivers a lazy seq. Demanding the next element of the lazy seq results into a line being read from the reader.

Number 3: how to write to a new file.

```
(use 'clojure.java.io)
(with-open [wrtr (writer "/tmp/test.txt")]
  (.write wrtr "Line to be written"))
```

Again, `with-open` takes care that the `BufferedWriter` is closed at the end of the body. Writer coerces a string into a `BufferedWriter`, that you use use via java interop: `(.write wrtr "something").`

You could also use `spit`, the opposite of `slurp`:

```
(spit "/tmp/test.txt" "Line to be written")
```

Number 4: append a line to an existing file.

```
(use 'clojure.java.io)
(with-open [wrtr (writer "/tmp/test.txt" :append true)]
  (.write wrtr "Line to be appended"))
```

Same as above, but now with append option.

Or again with `spit`, the opposite of `slurp`:

```
(spit "/tmp/test.txt" "Line to be written" :append true)
```

PS: To be more explicit about the fact that you are reading and writing to a File and not something else, you could first create a File object and then coerce it into a `BufferedReader` or Writer:

```
(reader (file "/tmp/test.txt"))
;; or
(writer (file "tmp/test.txt"))
```

The file function is also in clojure.java.io.

PS2: Sometimes it's handy to be able to see what the current directory (so ".") is. You can get the absolute path in two ways:

```
(System/getProperty "user.dir")
```

or

```
(-> (java.io.File. ".") .getAbsolutePath)
```

edited Jan 5 '15 at 21:51
Dave Liepmann
814 1 11 18

answered Oct 13 '11 at 16:59
Michiel Borkent
17.5k 13 58 103

1 Thank you very very much for your detailed answer. I'm glad to get to know the recommended way of File IO (text file) in 1.3. There seems to have been some libraries about File IO (clojure.contrb.io, clojure.contrib.duck-streams and some examples directly using Java BufferedReader FileInputStream InputStreamReader) which made me more confusing. Moreover there is little information about Clojure 1.3 especially in Japanese (my natural language) Thank you. – jolly-san Oct 14 '11 at 0:51

Hi jolly-san, tnx for accepting my answer! For your information, clojure.contrib.duck-streams is now deprecated. This possibly adds to the confusion. – Michiel Borkent Oct 14 '11 at 7:21

Very informative. Thanks. – octopusgrabbus Nov 14 '12 at 14:35

2 This has to do with lazyness. When you use the result of line-seq outside of the with-open, which happens when you print its result to the REPL, then the reader is already closed. A solution is to wrap the line-seq inside a doall, which forces evaluation immediately. `(with-open [rdr (reader "/tmp/test.txt")] (doall (line-seq rdr)))` – Michiel Borkent Oct 27 '13 at 9:27

1 Beautiful Answer! – Yavar Apr 22 '16 at 15:16

If the file fits into memory you can read and write it with slurp and spit:

```
(def s (slurp "filename.txt"))
```

(s now contains the content of a file as a string)

```
(spit "newfile.txt" s)
```

This creates newfile.txt if it doesnt exit and writes the file content. If you want to append to the file you can do

```
(spit "filename.txt" s :append true)
```

To read or write a file linewise you would use Java's reader and writer. They are wrapped in the namespace clojure.java.io:

```
(ns file.test
  (:require [clojure.java.io :as io]))

(let [wrtr (io/writer "test.txt")]
  (.write wrtr "hello, world!\n")
  (.close wrtr))

(let [wrtr (io/writer "test.txt" :append true)]
  (.write wrtr "hello again!")
  (.close wrtr))

(let [rdr (io/reader "test.txt")]
  (println (.readLine rdr))
  (println (.readLine rdr)))
; "hello, world!"
; "hello again!"
```

Note that the difference between slurp/spit and the reader/writer examples is that the file remains open (in the let statements) in the latter and the reading and writing is buffered, thus more efficient when repeatedly reading from / writing to a file.

Here is more information: slurp spit clojure.java.io Java's BufferedReader Java's Writer

edited Oct 13 '11 at 16:43

answered Oct 13 '11 at 16:08
Paul
4,902 1 26 37

1 Thank you Paul. I could learn more by your codes and your comments which are clear in the point focusing on answering my question. Thank you very much. – jolly-san Oct 14 '11 at 0:55

Thanks for adding information on slightly lower-level methods not given in Michiel Borkent's (excellent) answer on best methods for typical cases. – Mars May 2 '14 at 17:43

@Mars Thanks. Actually I did answer this question first, but Michiel's answer has more structure and that seems to be very popular. – Paul May 3 '14 at 7:23

He does a good job with the usual cases, but you provided other information. That's why it's good that SE allows multiple answers. – Mars May 3 '14 at 21:27

Regarding question 2, one sometimes wants the stream of lines returned as a first-class object. To get this as a lazy sequence, and still have the file closed automatically on EOF, I used this function:

```
(use 'clojure.java.io)

(defn read-lines [filename]
  (let [rdr (reader filename)]
    (defn read-next-line []
      (if-let [line (.readLine rdr)]
       (cons line (lazy-seq (read-next-line)))
       (.close rdr)))
    (lazy-seq (read-next-line)))
)

(defn echo-file []
  (doseq [line (read-lines "myfile.txt")]
    (println line)))
```

answered Dec 10 '12 at 13:52
satyagraha
323 3 9

5 I don't think nesting `defn` is ideomatic Clojure. Your `read-next-line`, as far as I understand, is visible outside of your `read-lines` function. You might have used a `(let [read-next-line (fn [] ...))` instead. – kristianlm Oct 3 '13 at 16:48

This is how to read the whole file.

If the file is in the resource directory, you can do this :

```
(let [file-content-str (slurp (clojure.java.io/resource
"public/myfile.txt")])
```

remember to require/use clojure.java.io

edited Mar 23 '15 at 16:56
Dave Liepmann
814 1 11 18

answered Apr 22 '13 at 13:23
joshua
2,263 3 30 43