Clojure Goodness

Experience the Clojure programming language through code snippets

NOTE BOOK

Hubert A. Klein Ikkink

Clojure Goodness Notebook

Experience the Clojure programming language through code snippets

Hubert A. Klein Ikkink (mrhaki)

This book is for sale at http://leanpub.com/clojure-goodness-notebook

This version was published on 2023-06-29

Contents

Strings

Formatting With Java Format String

In Clojure we can format a string using Common Lisp format syntax or the Java format string syntax. In the post we will look at the how we can use the Java format string syntax. We must use the `format` function in the `clojure.core` namespace. The method delegates to the standard JDK `String#format` method. The first argument is a format string followed by one or more arguments that are used in the format string. We can look up the syntax of the format string in the Javadoc for the `java.util.Formatter` class.

In the following example code we use the `format` function with different format strings:

```
(ns mrhaki.string.format
  (:require [clojure.test :refer [is]])
  (:import (java.util Locale)))

;; Create new string with format string as template as first argument.
;; Following arguments are used to replace placeholders in the
;; format string.
;; Clojure will delegate to the java.lang.String#format method and
;; we can use all format string options that are defined for this method.
;; More details about the format string syntax can be found in
;; java.util.Formatter. In a REPL we can find the docs
;; with (javadoc java.util.Formatter).
(is (= "https://www.mrhaki.com/"
       (format "https://%s/" "www.mrhaki.com")))

;; Format string with argument index to refer to one argument twice.
(is (= "clojure CLOJURE"
       (format "%1$s %1$S" "clojure")))

;; Format string to define fixed result lenght of 10 characers
;; with padding to get the given length.
(is (= "   Clojure"
       (format "%10s" "Clojure")))

;; Default Locale is used to determine how locale specific
;; formats are applied. In the following example the default
;; decimal separator is . and group separator is , as specified
;; for the Canadian Locale.
(Locale/setDefault Locale/CANADA)
(is (= "Total: 42,000.00"
       (format "Total: %,.2f", 42000.0)))

(defn format-locale
  "Format a string using String/format with a Locale parameter"
  [locale fmt & args]
  (String/format locale fmt (to-array args)))
```

```
;; We can use a different Locale to apply different specific
;; locale formats. In the next example we use the Dutch Locale
;; and the decimal seperator is , and the group separator is ..
(is (= "Totaal: 42.000,00"
       (format-locale (Locale. "nl") "Totaal: %,.2f" 42000.0)))
```

Written with Clojure 1.10.1.

Original post written on October 10, 2020

Trimming Strings

In the `clojure.string` namespace we can find several useful function for working with strings. If we want to trim a string we can choose for the `trim`, `trial`, `trimr` and `trim-newline` functions. To trim all characters before a string we must use the `triml` function. To remove all space characters after a string we use `trimr`. To remove space characters both before and after a string we can use the `trim` function. Finally if we only want to remove the newline and/or return characters we use the `trim-newline` function.

In the following example we use the different trim functions on strings:

```
(ns mrhaki.sample
  (:require [clojure.test :refer [is]]
            [clojure.string :refer [trim triml trimr trim-newline]]))

;; The trim function removes spaces before and after the string.
(is (= "mrhaki" (trim " mrhaki ")))
;; Tabs are also trimmed.
(is (= "mrhaki" (trim "\t mrhaki ")))
;; Return and/or newline characters are also trimmed.
(is (= "mrhaki" (trim "\tmrhaki \r\n")))
;; Character literals that should be trimmed are trimmed.
(is (= "mrhaki" (trim (str \tab \space " mrhaki " \newline))))

;; The triml function removes spaces before the string (trim left).
(is (= "mrhaki " (triml " mrhaki ")))
(is (= "mrhaki " (triml "\t mrhaki ")))
(is (= "mrhaki " (triml "\nmrhaki ")))
(is (= "mrhaki " (triml (str \return \newline " mrhaki "))))

;; The trimr function removes spaces after the string (trim right).
(is (= " mrhaki" (trimr " mrhaki ")))
(is (= " mrhaki" (trimr " mrhaki\t")))
(is (= " mrhaki" (trimr (str " mrhaki " \newline))))

;; The trim-newline function removes only newline from string.
(is (= "mrhaki " (trim-newline (str "mrhaki " \newline))))
(is (= "mrhaki " (trim-newline (str "mrhaki " \return \newline))))
(is (= "mrhaki " (trim-newline "mrhaki \r\n")))
(is (= "mrhaki " (trim-newline "mrhaki ")))
```

Written with Clojure 1.10.1.

Original post written on January 3, 2020

Check Substring Is Part Of String

Sometimes we want to see if a string is part of another string. Or if the string value starts or ends with a certain string. In Clojure we can use the `includes?` function from the `clojure.string` namespace to check if a string is part of another string value. To see if a string starts with a certain value we can use the `starts-with?` function from the `clojure.string` namespace. And to check if a string ends with a given value we use `ends-with?` from the same namespace.

In the next example code we use these functions and also add a `matches?` function to check if a string matches with a regular expression defined in a string:

```
(ns mrhaki.string.includes
  (:require [clojure.string :as str]
            [clojure.test :refer [is]]))

;; String to check.
(def s "Clojure is cool!")

;; Check if given value is part of the string.
(is (true? (str/includes? s "cool")))
(is (false? (str/includes? s "boring")))

;; Check string starts with given value.
(is (true? (str/starts-with? s "Clojure")))
(is (false? (str/starts-with? s "Groovy")))

;; Check string ends with given value.
(is (true? (str/ends-with? s "cool!")))

;; Helper function to see if string with regular expression
;; matches a given string value using java.lang.String#matches.
(defn matches?
  "Return true when string `re` with regular expression
  matches for value `s`, false otherwise."
  [^CharSequence s ^CharSequence re]
  (. s matches re))

(is (true? (matches? s ".*is.*")))
(is (false? (matches? s "cool")))
```

Written with Clojure 1.10.1.

Original post written on April 22, 2020

Splitting Strings

In Clojure we can use the `clojure.string/split` function to split a string, based on a regular expression, into a vector with string values. Optionally we can also specify a limit on the maximum number of returned string values we want. If we want to split a string based on the newline characters we can use the function `clojure.string/split-lines` that returns a vector where each element is a line from the original multi-line string.

The following example shows several usages of the `split` and `split-lines` functions:

```
(ns mrhaki.string.split
  (:require [clojure.string :as str]
            [clojure.test :refer [is are]]))

;; Sample string to split.
(def issue "CLJ-90210: Subject")

;; Split on - and : to get vector with string values.
(is (= ["CLJ" "90210" "Subject"] (str/split issue #"-|: ")))

;; The split function accepts a third argument that is
;; a limit on the number of splits that are returned.
(is (= ["CLJ" "90210" "Subject"]
       (str/split issue #"-|: " 0)
       (str/split issue #"-|: " 3)))

(is (= [issue] (str/split issue #"-|: " 1)))
(is (= ["CLJ" "90210: Subject"] (str/split issue #"-|: " 2)))


;; Multiline sample string to split per line and
;; the split each line.
(def itinerary "loc:LVG time:15h-16h activity:Binge-watching
loc:DNR time:18h-19h activity:Eating
loc:MBR time:23h-7h activity:Sleeping")

;; Using split-line function we get a vector
;; where each line is an element.
;; Then for each line we split on : and \s+ and
;; convert it to a map.
;; E.g. first line is
;; {"loc" "LVG" "time" "15h-16h" "activity" "Binge-watching"}
(def agenda (map #(apply hash-map (str/split % #":|\s+"))
                 (str/split-lines itinerary)))

(is (= "LVG" ((first agenda) "loc")))
(is (= "15h-16h" ((first agenda) "time")))
(is (= "Binge-watching" ((first agenda) "activity")))

(is (= "DNR" ((nth agenda 1) "loc")))
(is (= "18h-19h" ((nth agenda 1) "time")))
(is (= "Eating" ((nth agenda 1) "activity")))

(are [value m key] (= value ((last m) key))
                   "MBR" agenda "loc"
                   "23h-7h" agenda "time"
                   "Sleeping" agenda "activity")
```

Written with Clojure 1.10.1.

Original post written on April 1, 2020

Joining Elements in a Collection

We can use the `join` function from the `clojure.string` namespace to join elements from a collection into a string. We can optionally specify a separator that is used to separate each element in the string output. The separator is not used after the last element of the collection. If we don't specify a separator the elements are concatenated without separation. The string representation for each element in the collection is used in the joined end result.

In the following example code we see different usages of the `join` function:

```
(ns mrhaki.sample
  (:import [java.util Currency Locale])
  (:require [clojure.string :refer [join]]
            [clojure.test :refer [is]]))

;; Join without explicit separator simply concats values in collection.
(is (= "abc" (join ["a" "b" "c"])))

;; Join with separator uses separator between elements from collection
;; and omits the separator after the last element.
(is (= "a, b, c" (join ", " ["a" "b" "c"])))

;; Join works on multiple collection types,
;; because each collection is transformed to a seq.
(is (= "a::b::c" (join "::" #{"a" "b" "c"})))

;; Collection with non-strings is also returned as string.
;; The string representation of each element is used.
(is (= "0 1 2 3 4 5 6 7 8 9 10" (join " " (range 11))))
(is (= "https://www.mrhaki.com:443/,EUR" (join \, [(java.net.URL. "https" "www.mrhaki.com" 443 "/")
                                                    (Currency/getInstance (Locale. "nl" "NL"))])))

;; Nil values are ignored in the join results,
;; but separator is still used for nil element.
(is (= "Clojure--is cool--!" (join "-" ["Clojure" nil "is cool" nil "!"])))

;; Function query-params to transform a map structure with
;; keyword keys to URL request parameters.
(defn query-params
  "Return key/value pairs as HTTP request parameters separated by &.
   Each request parameter name and value is separated by =.
   E.g. {:q \"Clojure\" :max 10 :start 0 :format \"xml\"} is transformed
   to q=Clojure&max=10&start=0&format=xml."
  [params]
  (let [query-param (fn [[param-name param-value]] (join "=" [(name param-name) param-value]))]
    (join "&" (map query-param params))))

(is (= "q=Clojure&max=10&start=0&format=xml" (query-params {:q "Clojure" :max 10 :start 0 :format "xml\
"})))
```

Written with Clojure 1.10.1

Original post written on January 6, 2020

Replacing Characters In A String With escape Function

The `clojure.string` namespace contains a lot of useful functions to work with string values. The `escape` function can be used to replace characters in a string with another character. The function accepts as first argument the string value and the second argument is a map. The map has characters as key that need to be replaced followed by the value it is replaced with. For example the map `{\a 1 \b 2}` replaces the character `a` with `1` and the character `b` with `2`.

In the following example code we use the `escape` function in several cases:

```
(ns mrhaki.string.escape-string
  (:require [clojure.string :as str]
            [clojure.test :refer [is]]))

(is (= "I 10v3 C10jur3"
       (str/escape "I love Clojure" {\o 0 \e 3 \l 1})))

(is (= "mrHAKI"
       (str/escape "mrhaki" {\h "H" \a "A" \k "K" \i "I" \x "X"})))

(def html-escaping {(char 60) "&lt;" (char 62) "&gt;" (char 38) "&amp;"})
(is (= "&lt;h1&gt;Clojure &amp; Groovy rocks!&lt;/h1&gt;"
       (str/escape "<h1>Clojure & Groovy rocks!</h1>" html-escaping)))

(is (= "Special chars: \\t \\n"
       (str/escape "Special chars: \t \n" char-escape-string)))
```

Written with Clojure 1.10.1.

Original post written on July 8, 2020

Replacing Matching Values In String

We can search for a value in a string and replace it with another value using the `clojure.string/replace` function. The first parameter is the original string value that we want to replace parts of. The second parameter can be a string value or regular expression. The last parameter is the replacement value that can be a string value or a function that returns a string value. The function itself gets either a string argument if the match has no nested groups (when match is a regular expression) or a vector with a complete match followed by the nested groups when the match has nested groups.

In the following example we several invocation of the `clojure.string/replace` function with different arguments:

```
(ns mrhaki.string.replace
  (:require [clojure.string :as str]
            [clojure.test :refer [is]]))

;; Example string value to do replacements on.
(def s "Programming with Clojure is fun!")

;; Match argument can be a string value,
;; that gives same result as java.lang.String#replace method.
(is (= "Programming with Clojure is awesome!"
       (str/replace s "fun" "awesome")
       (.replace s "fun" "awesome")))


;; Match argument can also be regular expression pattern.
(is (= "Programming_with_Clojure_is_fun!"
       (str/replace s #"\s+" "_")))

;; When the regular expression pattern has groups
;; we can refer to them using $ followed by matched
;; group number, eg. $1 for the first group.
(is (= "Execution 1 took 200ms"
       (str/replace "run1=200ms" #"run(\d+)=(\d+ms)" "Execution $1 took $2")))


;; Replace argument can be a function.
;; Argument of the function is string of entire match
;; if there are no nested groups.
(is (= "[NOTE] [CAUTION]"
       (str/replace "[note] [caution]" #"\[\w+\]" #(.toUpperCase %))))

;; Otherwise if there are nested groups a vector is
;; used as argument for the replacment function
;; where the first argument is the
;; entire match followed by the nested groups.
(is (= "ABC def"
       (str/replace "abc DEF"
                    #"(\w+)(\s+)(\w+)"
                    #(str (.toUpperCase (% 1)) (% 2) (.toLowerCase (% 3))))))

;; By destructuring the vector argument
;; we can refer to the groups using a name.
(defn replacement
  [[_ execution time]]
  (let [seconds (/ (bigdec time) 1000)]
    (str "Execution " execution " took " seconds " seconds")))

(is (= "Execution 1 took 0.2 seconds"
       (str/replace "run1=200ms" #"run(\d+)=(\d+)ms" replacement)))
```

Written with Clojure 1.10.1.

Original post written on March 29, 2020