Package 'fastmatch'

July 2, 2014

Version 1.0-4

Title Fast match() function

Author Simon Urbanek <simon.urbanek@r-project.org>

Maintainer Simon Urbanek <simon.urbanek@r-project.org>

Description Package providing a fast match() replacement for cases
that require repeated look-ups. It is slightly faster that R's
built-in match() function on first match against a table, but
extremely fast on any subsequent lookup as it keeps the hash table in memory.

License GPL-2

URL http://www.rforge.net/fastmatch

Repository CRAN

Date/Publication 2012-01-21 10:22:24

NeedsCompilation yes

R topics documented:

`fmatch` *Fast match() replacement*

Description

`fmatch` is a faster version of the built-in `match`() function. It is slightly faster than the built-in version because it uses more specialized code, but in addition it retains the hash table within the table object such that it can be re-used, dramatically reducing the look-up time especially for large tables.

Although `fmatch` can be used separately, in general it is also safe to use: `match <- fmatch` since it is a drop-in replacement. Any cases not directly handled by `fmatch` are passed to `match` with a warning.

Usage

```
fmatch(x, table, nomatch = NA_integer_, incomparables = NULL)
```

Arguments

`x`	values to be matched
`table`	values to be matched against
`nomatch`	the value to be returned in the case when no match is found. It is coerced to `integer`.
`incomparables`	a vector of values that cannot be matched. Any value other than `NULL` will result in a fall-back to `match` without any speed gains.

Details

See `match` for the purpose and details of the `match` function. `fmatch` is a drop-in replacement for the `match` function with the focus on performance. `incomparables` are not supported by `fmatch` and will be passed down to `match`.

The first match against a table results in a hash table to be computed from the table. This table is then attached as the '.match.hash' attribute of the table so that it can be re-used on subsequent calls to `fmatch` with the same table.

The hashing algorithm used is the same as the `match` function in R, but it is re-implemented in a slight different way to improve its performance at the cost of supporting only a subset of types (integer, real and character). For any other types `fmatch` falls back to `match` (with a warning).

Value

A vector of the same length as `x` - see `match` for details.

Note

fmatch modifies the table by attaching an attribute to it. It is expected that the values will not change unless that attribute is dropped. Under normal circumstances this should not have any effect from user's point of view, but there is a theoretical chance of the cache being out of sync with the table in case the table is modified directly (e.g. by some C code) without removing attributes.

Also fmatch does not convert to a common encoding so strings with different representation in two encodings don't match.

See Also

match

Examples

```
# some random speed comparison examples:
# first use integer matching
x = as.integer(rnorm(1e6) * 1000000)
s = 1:100
# the first call to fmatch is comparable to match
system.time(fmatch(s,x))
# but the subsequent calls take no time!
system.time(fmatch(s,x))
system.time(fmatch(-50:50,x))
system.time(fmatch(-5000:5000,x))
# here is the speed of match for comparison
system.time(base::match(s, x))
# the results should be identical
identical(base::match(s, x), fmatch(s, x))

# next, match a factor against the table
# this will require both x and the factor
# to be cast to strings
s=factor(c("1","1","2","foo","3",NA))
# because the casting will have to allocate a string
# cache in R, we run a dummy conversion to take
# that out of the equation
dummy = as.character(x)
# now we can run the speed tests
system.time(fmatch(s, x))
system.time(fmatch(s, x))
# the cache is still valid for string matches as well
system.time(fmatch(c("foo","bar","1","2"),x))
# now back to match
system.time(base::match(s, x))
identical(base::match(s, x), fmatch(s, x))

# finally, some reals to match
y = rnorm(1e6)
s = c(y[sample(length(y), 100)], 123.567, NA, NaN)
system.time(fmatch(s, y))
system.time(fmatch(s, y))
```

```
system.time(fmatch(s, y))
system.time(base::match(s, y))
identical(base::match(s, y), fmatch(s, y))

# this used to fail before 0.1-2 since nomatch was ignored
identical(base::match(4L, 1:3, nomatch=0), fmatch(4L, 1:3, nomatch=0))
```

Index