Package ‘getDTeval’

September 13, 2020

Title Implements Programmatically Designed Coding Statements without Compromising on the Runtime Performance

Version 0.0.1

Depends R (¿= 3.1.0)

Description The getDTeval() function facilitates the translation of the original coding statement to an optimized form for improved runtime efficiency without compromising on the programmatic coding design.
The function can either provide a translation of the coding statement, directly evaluate the translation to return a coding result, or provide both of these outputs.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, dplyr, formulaic, testthat (¿= 2.1.0)

Imports data.table, microbenchmark, stats, utils

VignetteBuilder knitr

NeedsCompilation no

Author David Shilane [aut],
Mayur Bansal [ctb, cre],
Anderson Nelson [ctb],
Caffrey Lee [ctb],
Zichen Huang [ctb]

Maintainer Mayur Bansal <`mb4511@columbia.edu`>

R topics documented:

`benchmark.getDTeval` *benchmark.getDTeval*

Description

Performs a benchmarking experiment for data.table coding statements that use get() or eval() for programmatic designs. The a) original statement is compared to b) passing the original statement through getDTeval and also to c) an optimized coding statement. The results can demonstrate the overall improvement of using the coding translations offered by getDTeval()

Usage

```
benchmark.getDTeval(
  the.statement,
  times = 30,
  seed = 47,
  envir = .GlobalEnv,
  ...
)
```

Arguments

`the.statement`	refers to the original coding statement which needs to be translated to an optimized form. This value may be entered as either a character value or as an expression.
`times`	The number of iterations to run the benchmarking experiment
`seed`	an integer value specifying the seed of the pseudorandom number generator.
`envir`	The environment in which the calculation takes place, with the global environment .GlobalEnv set as the default.
`...`	provision for additonal arguments

Examples

```
# Using benchmark.getDteval to compare runtime performance of original coding statement, optimized st
dat<-formulaic::snack.dat
age.name<-'Age'
benchmark.getDTeval(the.statement = 'dat[,.(mean_age=mean(Age))]', times = 50, seed = 282)
```

`function.ending.point` *function.ending.point*

Description

An Internal function to return the ending index

Usage

```
function.ending.point(all.chars, beginning.index, ...)
```

Arguments

`all.chars`	all the characters of the statement
`beginning.index`	specifies the index of the first character
`...`	provision for additional arguments

`getDTeval` *getDTeval*

Description

The getDTeval() function facilitates the translation of the original coding statement to an optimized form for improved runtime efficiency without compromising on the programmatic coding design. The function can either provide a translation of the coding statement, directly evaluate the translation to return a coding result, or provide both of these outputs

Usage

```
getDTeval(
  the.statement,
  return.as = "result",
  coding.statements.as = "character",
  eval.type = "optimized",
  envir = .GlobalEnv,
  ...
)
```

Arguments

`the.statement`	refers to the original coding statement which needs to be translated to an optimized form. This value may be entered as either a character value or as an expression.
`return.as`	refers to the mode of output. It could return the results as a coding statement (return.as = ”code”), an evaluated coding result (return.as = ”result”, which is the default value), or a combination of both (return.as = ”all”).
`coding.statements.as`	determines whether the coding statements provided as outputs are returned as expression objects (return.as = ”expression”) or as character values (return.as = ”character”, which is the default).
`eval.type`	a character value stating whether the coding statement should be evaluated in its current form (eval.type = ”as.is”) or have its called to get() and eval() translated (eval.type = ”optimized”, the default setting).
`envir`	Specify the environment for the required function. .GlobalEnv is set as default
`...`	provision for additional arguments

Examples

```
# Using getDTeval to calculate mean age
dat<-formulaic::snack.dat
age.name<-'Age'
getDTeval(the.statement = 'dat[,.(mean_age=mean(get(age.name)))]',return.as = 'result')
```

`translate.fn.calls` *translate.fn.calls*

Description

Internal Function that translates programmatic designs into optimized coding statements for faster calculations

Usage

```
translate.fn.calls(
  the.statement,
  function.name = "get(",
  envir = .GlobalEnv,
  ...
)
```

Arguments

`the.statement`	original coding statement to perform the required calculation. Must be provided as a character value.
`function.name`	Name of the function to be translated to an optimized form. Parameter values should be either 'get(' or 'eval('. 'get(' is set as default
`envir`	Specify the environment for the required function. .GlobalEnv is set as default
`...`	provision for additional arguments