Generate global option function

Zuguang Gu <z.gu@dkfz.de>

September 15, 2014

Global option function such as `options` and `par` can provide a way to control global settings. Here the *GlobalOptions* package can generate a function which takes care of such global settings.

The most simple use is to generate an option function by `setGlobalOptions`:

```
> library(GlobalOptions)
> foo.options = setGlobalOptions(
+     a = 1,
+     b = "text"
+ )
```

The returned value `foo.options` is an option function which can be used to get or set options:

```
> foo.options()

$a
[1] 1

$b
[1] "text"

> foo.options("a")

[1] 1

> op = foo.options()
> op

$a
[1] 1

$b
[1] "text"

> foo.options(a = 2, b = "new text")
> foo.options()

$a
[1] 2

$b
[1] "new text"

> foo.options(op)
> foo.options()

$a
[1] 1

$b
[1] "text"
```

`foo.options` generated by `setGlobalOptions` will contain an argument `RESET` which is used to reset the options to the default:

```
> foo.options(a = 2, b = "new text")
> foo.options(RESET = TRUE)
> foo.options()

$a
[1] 1

$b
[1] "text"
```

If values for options are set as lists, more close controls can be customized. There are two basic conditions that are used to check the input option values:

```
> foo.options = setGlobalOptions(
+     a = list(.value = 1,
+              .length = c(1, 3),
+              .class = "numeric")
+ )
```

In above code, `.value` is the default value for the option `a`, the length of the value is controlled by `.length` and the length should be either 1 or 3, and the class of the value should be `numeric`. If the input value does not fit the criterion, there will be an error. The value of `.length` or `.class` is a vector and the checking will be passed if one of the value fits user's input value.

```
> foo.options(a = 1:2)

Error in foo.options(a = 1:2) : Length of 'a' should be one of 1, 3

> foo.options(a = "text")

Error in foo.options(a = "text") :
  Class of 'a' should be one of 'numeric'.
```

User can set the value as read-only and modifying such option will cause an error.

```
> foo.options = setGlobalOptions(
+     a = list(.value = 1,
+              .read.only = TRUE)
+ )
> foo.options(a = 2)

Error in foo.options(a = 2) : 'a' is a read-only option.
```

There is also a pre-defined argument `READ.ONLY` exported with `foo.options` which controls whether to return only the read-only options or not.

```
> foo.options = setGlobalOptions(
+     a = list(.value = 1,
+              .read.only = TRUE),
+     b = 2
+ )
> foo.options()

$a
[1] 1

$b
[1] 2
```

```
> foo.options(READ.ONLY = TRUE)

$a
[1] 1

> foo.options(READ.ONLY = FALSE)

$b
[1] 2
```

Validation of the option values can be set by `.validate`. The value should be a function. The input of the validation function is the input option value and the function should only return a logical value.

```
> foo.options = setGlobalOptions(
+     a = list(.value = 1,
+              .validate = function(x) x > 0 && x < 10)
+ )
> foo.options(a = 20)

Error in foo.options(a = 20) : Your option is invalid.
```

Filtering on the option values can be set by `.filter`. This is useful when the input option value is not valid but users only want to change the value without throwing errors. For example, there is an option to control whether to print messages or not and it should be set to `TRUE` or `FALSE`. Anyway, users may set some other type of values such as `NULL` or `NA`. In this case, non-`TRUE` values can be converted to logical values inside `.filter`. Similar as `.validate`, the input value for filter function is the input option value, and it should return a filtered option value.

```
> foo.options = setGlobalOptions(
+     verbose =
+         list(.value = TRUE,
+              .filter = function(x) {
+                  if(is.null(x)) {
+                      return(FALSE)
+                  } else if(is.na(x)) {
+                      return(FALSE)
+                  } else {
+                      return(x)
+                  }
+               })
+ )
> foo.options(verbose = FALSE); foo.options("verbose")

[1] FALSE

> foo.options(verbose = NA); foo.options("verbose")

[1] FALSE

> foo.options(verbose = NULL); foo.options("verbose")

[1] FALSE
```

The input option value can be set as dynamic by setting it as a function. When the option value is set as a function, it will be executed when querying the option. In the following example, the `prefix` option corresponds to the prefix of some log messages.

```
> foo.options = setGlobalOptions(
+     prefix = ""
+ )
> foo.options(prefix = function() paste("[", Sys.time(), "] ", sep = " "))
> foo.options("prefix")
```

```
[1] "[ 2014-09-15 23:09:21 ] "
```

```
> Sys.sleep(2)
> foo.options("prefix")
```

```
[1] "[ 2014-09-15 23:09:21 ] "
```

If the value of the option is really a function and users don't want to execute it, just set `.class` to `function`, then the function will be treated as a simple value.

```
> foo.options = setGlobalOptions(
+     test = list(.value = function(x1, x2) t.test(x1, x2)$p.value,
+                 .class = "function")
+ )
> foo.options(test = function(x1, x2) cor.test(x1, x2)$p.value)
> foo.options("test")
```

```
function(x1, x2) cor.test(x1, x2)$p.value
```

The self-defined function (*i.e.* value function, validation function or filter function) is applied per-option. But if it relies on other option values, there is pre-defined variable `OPT` which is a list containing values for all options. In following example, default value of option `b` is two times of the value of option `a`.

```
> foo.options = setGlobalOptions(
+     a = list(.value = 1),
+     b = list(.value = function() 2 * OPT$a)
+ )
> foo.options("b")
```

```
[1] 2
```

And in the second example, sign of `b` should be as same as sign of `a`.

```
> foo.options = setGlobalOptions(
+     a = list(.value = 1),
+     b = list(.value = 2,
+              .validate = function(x) {
+                  if(OPT$a > 0) x > 0
+                  else x < 0
+              },
+              .filter = function(x) {
+                  x + OPT$a
+              })
+ )
> foo.options("b")
```

```
[1] 2
```

```
> foo.options(a = 1, b = -1)
```

```
Error in foo.options(a = 1, b = -1) : Your option is invalid.
```

Global options are stored in a private environment. Each time when generating a option function, there will be a new environment created. Thus global options will not conflict if they come from different option functions.

```
> foo.options1 = setGlobalOptions(
+     a = list(.value = 1)
+ )
> foo.options2 = setGlobalOptions(
+     a = list(.value = 1)
+ )
> foo.options1(a = 2)
> foo.options1("a")
```

```
[1] 2
```

```
> foo.options2("a")
```

```
[1] 1
```

The option function generated by `setGlobalOptions` contains three arguments: `...`, `RESET` and `READ.ONLY`. If you want to put the option function into a package, remember to document all three arguments:

```
> args(foo.options)
```

```
function (..., RESET = FALSE, READ.ONLY = NULL)
NULL
```

The final and the most important thing is the check by `.class`, `.length`, `.validate`, `.filter` will not be applied on default values when calling `setGlobalOptions`.