Package 'compiler'

September 14, 2016

Version 3.3.1

Priority base

Title The R Compiler Package

Author Luke Tierney <luke-tierney@uiowa.edu>

Maintainer R Core Team <R-core@r-project.org>

Description Byte code compiler for R.

License Part of R 3.3.1

R topics documented:

`compile` *Byte Code Compiler*

Description

これらの関数は R に対するバイトコードコンパイルへのインタフェイスを提供する.

Usage

```
cmpfun(f, options = NULL)
compile(e, env = .GlobalEnv, options = NULL)
cmpfile(infile, outfile, ascii = FALSE, env = .GlobalEnv,
        verbose = FALSE, options = NULL)
loadcmp(file, envir = .GlobalEnv, chdir = FALSE)
disassemble(code)
enableJIT(level)
compilePKGS(enable)
getCompilerOption(name, options)
setCompilerOptions(...)
```

Arguments

f	クロージャ.
options	名前付きコンパイラオプションのリスト.
env	コンパイルのためのトップレベル環境.
file,infile,outfile	パス名；出力ファイルの既定値は入力ファイルの拡張子を .Rc に変えたものである.
ascii	論理値；コンパイル済みファイルはアスキー書式で保存すべきか?
verbose	論理値；コンパイラーは途中経過を表示すべきか.
envir	ロードされた表現式をそのなかで評価する環境.
chdir	論理値；評価前にディレクトリを変更すべきか?
code	バイトコード表現式かコンパイル済みクロージャ.
e	コンパイルする表現式.
level	整数；使用する JIT レベル.
enable	論理値；もし TRUE ならばパッケージのコンパイルを可能にする.
name	文字列；返すべきオプションの名前.
...	設定する名前付きコンパイルオプション.

Details

関数 cmpfun はクロージャの本体をコンパイルし同じ形式引数部とコンパイル済みの本体表現式に置き換えられた本体を持つ新しいクロージャを返す.

compile は表現式をバイトコードオブジェクトにコンパイルする；オブジェクトはそれから eval で評価できる.

cmpfile は infile 中の表現式を構文解析し，それらをコンパイルし，それからコンパイル済みの表現式を outfile に書き込む．もし outfile が与えられていないと，それは infile の拡張子を .Rc に変換又は付加したものになる.

loadcmp はコンパイルされたファイルをロードするのに使われる．これは sys.source に似ているが，その既定の環境が基礎環境ではなく大局的環境であることが異なる.

disassemble はコードのプリント形式の表現を作り，何が行われるのかのヒントを与えるのに役に立つかもしれない.

enableJIT はジャストインタイム(JIT)・コンパイルを有効にしたり無効にしたりする．引数が 0 ならば JIT は無効になる．もし enable が 1 ならばクロージャはその最初の使用の前にコンパイルされる．もし enable が 2 ならば，加えてクロージャはそれらが複製される前にコンパイルされる (lattice のようなリスト中にクロージャを保管するパッケージに対して役に立つ)．もし enable が 3 ならば更に全てのループが実行前にコンパイルされる．レベル 3 の JIT は最適化レベル 2 か 3 を必要とする． JIT は又環境変数 R_ENABLE_JIT をこれらの値の一つに設定して R を開始しても有効に出来る． enableJIT を負の値で呼び出すと現在の JIT レベルを返す.

compilePKGS はパッケージをインストールする際にコンパイルすることを有効にしたり無効にしたりする．コンパイルは関数が遅延ロードのデータベースに書き込まれるときに行われるため，これはパッケージが遅延ロードを使っている必要がある．これは又環境変数 R_COMPILE_PKGS を正の整数値に設定して R を開始しても有効に出来る.

現在コンパイラーは様々な事柄について警告する．これは cat を使ってメッセージをプリントする．いつかはこれは条件付き処理機構を使うべきである.

options 引数はコンパイラの操作を制御できる．現在三つのオプションがある：optimize, suppressAll そして suppressUndefined である． optimize は最適化のレベルを指定し， 0 から 3 までの整数である． suppressAll はスカラの論理値でなければならない；もし TRUE ならメッセージは一切表示されない． suppressUndefined が TRUE なら未定義の変数に関するメッセージが抑制され，もしくはそれはメッセージの表示を抑制する変数名のベクトルでも良い．

getCompilerOption は指定されたオプションの値を返す． options 引数中に値が提供されない限り既定値が返される； options 引数は主に内部用である． setCompilerOption は既定のオプションの値を設定する．それは以前の値の名前付きリストを返す．

コンパイラをバイトコードコンパイラと呼ぶのは実際には不適切である：コードオブジェクトの外部表現は現在 int オペランドを使っており， gcc でコンパイルすると内部表現は実際はバイトコードではなくスレッド化されたコードである．

Author(s)

Luke Tierney

Examples

```
# 簡単な例
f <- function(x) x+1
fc <- cmpfun(f)
fc(2)
disassemble(fc)

# lapply の古い R バージョン
la1 <- function(X, FUN, ...) {
    FUN <- match.fun(FUN)
    if (!is.list(X))
X <- as.list(X)
    rval <- vector("list", length(X))
    for(i in seq(along = X))
rval[i] <- list(FUN(X[[i]], ...))
    names(rval) <- names(X)   # `names' を保存!
    return(rval)
}
# 少し変更
la2 <- function(X, FUN, ...) {
    FUN <- match.fun(FUN)
    if (!is.list(X))
X <- as.list(X)
    rval <- vector("list", length(X))
    for(i in seq(along = X)) {
        v <- FUN(X[[i]], ...)
        if (is.null(v)) rval[i] <- list(v)
        else rval[[i]] <- v
    }
    names(rval) <- names(X)   # `names' を保存!
    return(rval)
}
# コンパイルされたバージョン
la1c <- cmpfun(la1)
la2c <- cmpfun(la2)
# あるタイミング
x <- 1:10
```

```
y <- 1:100

system.time(for (i in 1:10000) lapply(x, is.null))
system.time(for (i in 1:10000) la1(x, is.null))
system.time(for (i in 1:10000) la1c(x, is.null))
system.time(for (i in 1:10000) la2(x, is.null))
system.time(for (i in 1:10000) la2c(x, is.null))
system.time(for (i in 1:1000) lapply(y, is.null))
system.time(for (i in 1:1000) la1(y, is.null))
system.time(for (i in 1:1000) la1c(y, is.null))
system.time(for (i in 1:1000) la2(y, is.null))
system.time(for (i in 1:1000) la2c(y, is.null))
```

Index