## 18.3 在RStudio中制作R包

上文中提到，在Rstudio提供了关于R语言的功能强大的集成开发环境，Rstudio提供了完全可视化的R包制作接口，上述的R包描述性文件均可在Rstudio中直接编辑，但是这一过程所遵从的基本步骤是与上述在cmd命令行中进行的操作是一致的。

下面将演示如何利用RStudio制作上文中的hello-package.

**1 创建一个工程hello**

打开Rstudio，利用左上方的菜单，Rstudio>File>New Project，打开新项目。选择New Directory为待创建的R包建立新的工作路径，选择R package，如图18-5所示。

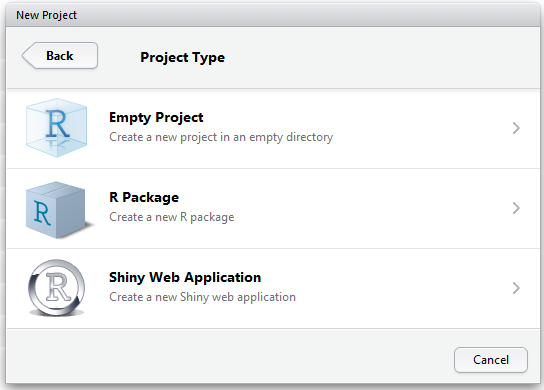


图18-5 创建新项目

在接下来的操作页面中输入与R包有关的信息，如包名，源文件链接，以及创建R包的工作路径，如图18-6所示。

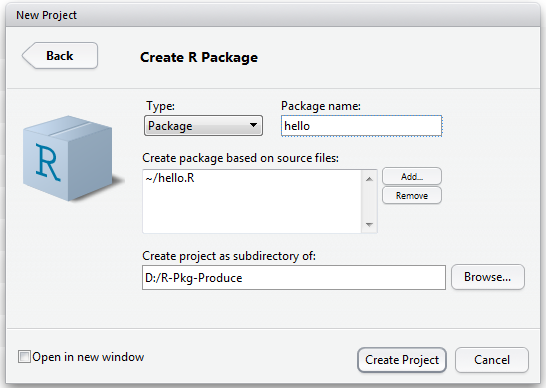


图18-6 输入相关信息

填选上述信息之后，点击“Creat Project”，在右下角展开如下工作页面，如图18-7所示：

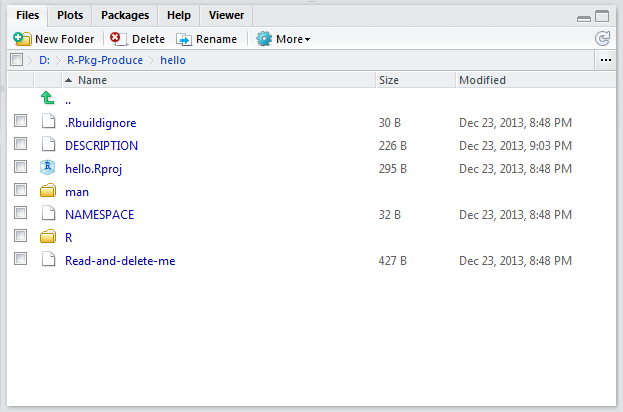


图18-7 创建项目

也可以通过如下命令创建R包：

devtools::create("path/to/package/pkgname")

这两种方法都可行并且生成了包含以下3部分最基本可使用的R包：

1. .R/
2. DESCRIPTION
3. NAMESPACE

生成的R包中也会包含一个RStudio项目文件,hello.Rproj，方便R包在RStudio中的使用。最好不要使用package.skeleton()来创建R包，因为那样做会给你带来更多的工作量，诸如修改一些必要的文件等。

**2 编辑包生成文件**

**2.1 code(R/)**

**组织函数**

最好将函数封装到文件中，形成良好的风格，此外脚本中的函数和包中的函数是有区别的。同时可以按照自己的意愿来组织函数，但是两个极端情况是不宜采纳的：一个是不要把所有的函数都放进一个文件中，也不要一个文件只包含一个函数。另外文件的名字应该是有含义的，以.R结尾，并且R/中不能有子文件夹。

# *Good*

fit\_models.R

utility\_functions.R

*# Bad*

foo.r

stuff.r

**代码风格**

有良好的代码风格如同使用正确的标点符号，这会使代码编写和阅读更快捷，特别是当你和其他人合作编写代码时。由谢益辉编写的formatR包，可以很快使自己的代码变得整洁。在使用之前，最好先阅读使用教程：

**install.packages**("formatR")

formatR::**tidy\_dir**("R")

或者使用由Jim Hester编写的lintr包，这个包并不能自动修改错误，但是可以提醒你问题的出处：

**install.packages**("lintr")

lintr::**lint\_package**()

**2.2 DESCRIPTION**

每个包都必须有一个DESCRIPTION文件（RStudio和devtools将任何含有DESCRIPTION的文件都视为包），通过devtools::create("mypackage")可以自动添加一个该文件，后续可以进行修改：

**Package:** mypackage

**Title:** What The Package Does (one line, title case required)

**Version:** 0.1

**Authors@R:** person("First", "Last", email = "first.last@example.com",

role = c("aut", "cre"))

**Description:** What the package does (one paragraph)

**Depends:** R (>= 3.1.0)

**License:** What license is it under?

**LazyData:** true

可直接点选对应的文件，打开编辑窗口编辑文件内容，文件的格式要求与前文一致，如图18-8所示。

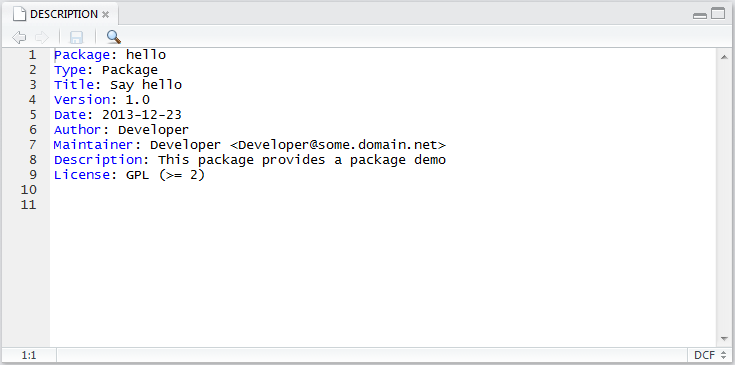


图18-8 编辑文件内容

DESCRIPTION文件使用的是DCF格式，the Debian control format，每行都包含一个域名和值，由冒号分隔。当值占据多行时需要缩进，一般是4个空格。

**Description:** The description of a package is usually long,

spanning multiple lines. The second and subsequent lines

should be indented, usually with four spaces.

**(1) 标题和概述（Title、Description）**

Title ：对包的一句话描述，经常在包列表中显示。应该是纯文本而非标记，大写，不以句号结尾。尽量简短，列表中title经常被截断到第65个字符。

Description ：比title更加详细，可以多行，但是限制在一个自然段内，每行不超过80个字符，而且要缩进4个空格。

**(2) 作者（Author）**

每个包至少有一个作者和维护者（他们可以是同一个人）。当问题出现的时候，可以使用Authors@R，这个域是可执行的R 代码，例如：

**Authors@R:** person("Hadley", "Wickham", email = "hadley@rstudio.com",

role = c("aut", "cre"))

**Authors@R:** c(

person("Hadley", "Wickham", email = "hadley@rstudio.com", role = "cre"),

person("Winston", "Chang", email = "winston@rstudio.com", role = "aut"))

除了邮箱地址，也可以添加URL（多个URL用逗号隔开）和BugReports：

**URL:** http://yihui.name/knitr/

**BugReports:** https://github.com/yihui/knitr/issues

**(3) Dependencies**

添加Imports和Suggests的方法是使用：devtools::use\_package()，而且自动添加到DESCRIPTION中正确的位置：

devtools::**use\_package**("dplyr") *# Defaults to imports*

*#> Adding dplyr to Imports*

*#> Refer to functions with dplyr::fun()*

devtools::**use\_package**("dplyr", "Suggests")

*#> Adding dplyr to Suggests*

*#> Use requireNamespace("dplyr", quietly = TRUE) to test if package is*

*#> installed, then use dplyr::fun() to refer to functions.*

如果需要特定版本的包，只需在包的名字后面用括号注明版本即可：

**Imports:**

ggvis (>= 0.2),

dplyr (>= 0.3.0.1)

**Suggests:**

MASS (>= 7.3.0)

**2.3 帮助文档（man/）**

文档有很多种形式，这里主要是? or help()得到的帮助文档，通过roxygen2自动生成特定的形式并写入.Rd文件中。

**基本流程**

(1) 在.R文件中直接添加roxygen comments。

(2) 运行devtools::document()将roxygen comments转换成.R文件(其中devtools::document()调用了roxygen2::roxygenise())。

(3) “?”预览文档。

Roxygen comments以#’开头，区别于一般注释，例如：

*#' Add together two numbers.*

*#'*

*#' @param x A number.*

*#' @param y A number.*

*#' @return The sum of \code{x} and \code{y}.*

*#' @examples#' add(1, 1)*

*#' add(10, 1)*

add <- function(x, y) {

x + y

}

运行devtools::document()将会生成如下的man/add.Rd说明文件：

% Generated by roxygen2 (4.0.0): do not edit by hand

\name{add}

\alias{add}

\title{Add together two numbers}

\usage{

add(x, y)

}

\arguments{

\item{x}{A number}

\item{y}{A number}

}

\value{

The sum of \code{x} and \code{y}

}

\description{

Add together two numbers

}

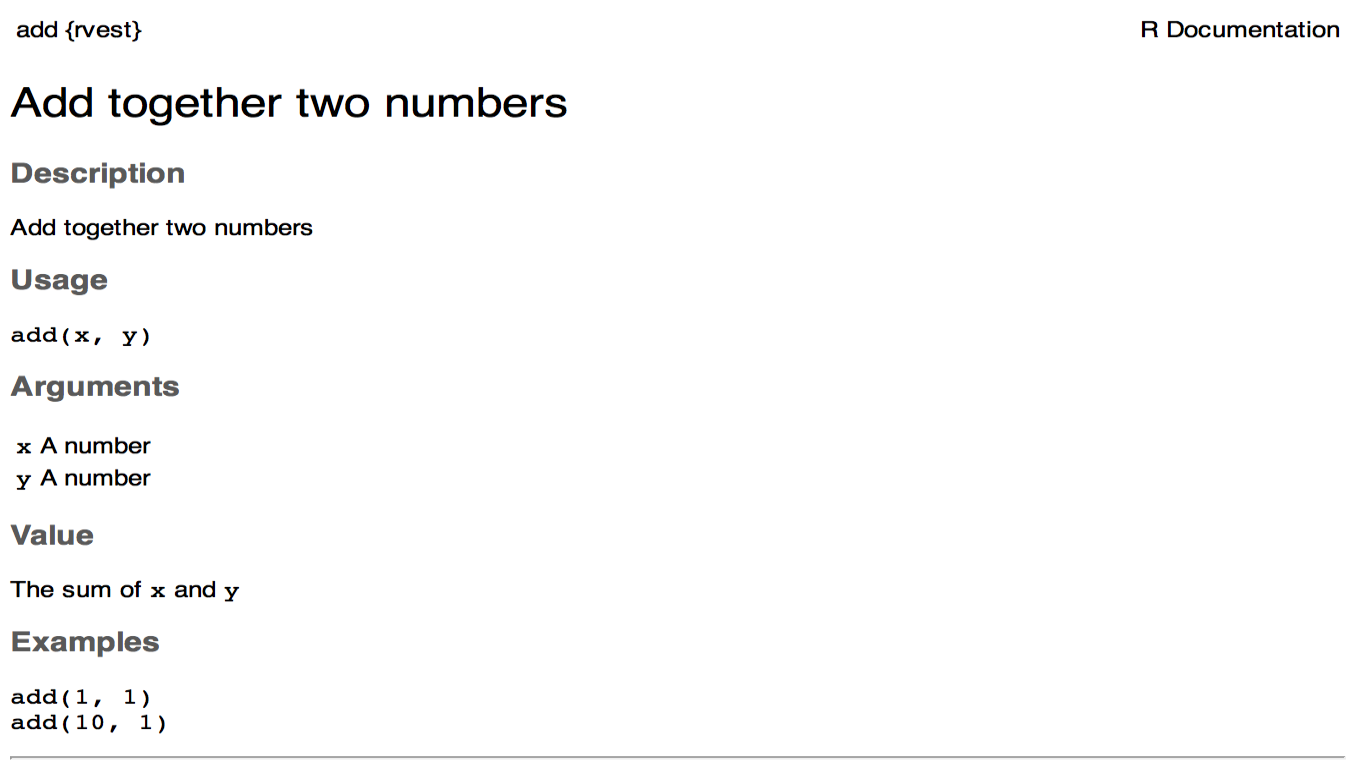
\examples{

add(1, 1)

add(10, 1)

}

运行?add或者help("add")：



**函数说明**

大部分函数说明都有三个标签:@param、@examples和@return

以sum函数的说明文档为例：

*#' Sum of vector elements.*

*#'*

*#' \code{sum} returns the sum of all the values present in its arguments.*

*#'*

*#' This is a generic function: methods can be defined for it directly*

*#' or via the \code{\link{Summary}} group generic. For this to work properly,*

*#' the arguments \code{...} should be unnamed, and dispatch is on the*

*#' first argument.*

*#'*

*#' @param ... Numeric, complex, or logical vectors.*

*#' @param na.rm A logical scalar. Should missing values (including NaN)*

*#' be removed?*

*#' @return If all inputs are integer and logical, then the output*

*#' will be an integer. If integer overflow*

*#' \url{http://en.wikipedia.org/wiki/Integer\_overflow} occurs, the output*

*#' will be NA with a warning. Otherwise it will be a length-one numeric or*

*#' complex vector.*

*#'*

*#' Zero-length vectors have sum 0 by definition. See*

*#' \url{http://en.wikipedia.org/wiki/Empty\_sum} for more details.*

*#' @examples*

*#' sum(1:10)*

*#' sum(1:5, 6:10)*

*#' sum(F, F, F, T, T)*

*#'*

*#' sum(.Machine$integer.max, 1L)*

*#' sum(.Machine$integer.max, 1)*

*#'*

*#' \dontrun{#' sum("a")*

*#' }*

sum <- function(..., na.rm = TRUE) {}

**数据说明**

数据集说明和函数说明只有微小的差别，数据集说明并不是对数据直接进行说明，而是对数据集的名字进行说明，同时保存在R/文件夹中。例如diamonds 数据集的说明如下：

*#' Prices of 50,000 round cut diamonds.*

*#'*

*#' A dataset containing the prices and other attributes of almost 54,000*

*#' diamonds.*

*#'*

*#' @format A data frame with 53940 rows and 10 variables:*

*#' \describe{*

*#' \item{price}{price, in US dollars}*

*#' \item{carat}{weight of the diamond, in carats}*

*#' ...*

*#' }*

*#' @source \url{http://www.diamondse.info/}*

"diamonds"

数据集说明有两个重要的标签：

@format：对数据有整体的概况，包括定义数据框以及每个变量。

@source：对数据的来源进行详细的说明，通常是\url{ }。

注意：数据集不用@export标签。

**包说明**

可以使用roxygen生成整个包的帮助文档，用来描述包中的重要部分。手动编辑@docType package和@name<package-name>，另外@section这个标签可以将文档按类分块：

*#' foo: A package for computating the notorious bar statistic.*

*#'*

*#' The foo package provides three categories of important functions:*

*#' foo, bar and baz.*

*#'*

*#' @section Foo functions:*

*#' The foo functions ...*

*#'*

*#' @docType package*

*#' @name foo*

NULL

将这个文件保存为<package-name>.R，最后通过package?<package-name>可以查看相应包的帮助文档。

**2.4 命名空间（Namespace）**

使用roxygen2生成命名空间和使用roxygen2生成函数说明文档类似，以testthat包中NAMESPACE文件为例：

# Generated by roxygen2 (4.0.2): do not edit by hand

S3method(as.character,expectation)

S3method(compare,character)

export(auto\_test)

export(auto\_test\_package)

export(colourise)

export(context)

exportClasses(ListReporter)

exportClasses(MinimalReporter)

importFrom(methods,setRefClass)

useDynLib(testthat,duplicate\_)

useDynLib(testthat,reassign\_function)

**基本流程**

1. 在.R文件中添加roxygen comments。
2. 运行devtools::document()将roxygen comments转换成.Rd文件。
3. 查看namespace中函数的导入导出是否正确。

**导出(Exports)**

如果函数可在包外使用，就必须声明该函数可以导出，并且只需在roxygen块中添加 @export 标签，例如：

*#' @export*

foo <- function(x, y, z) {

...

}

这将会产生export(),exportMethods()、xportClass()或者S3method()，这取决于目标对象的类型。

**导入(Imports)**

命名空间也可以控制哪些外部函数可以被包引用，这样就不必使用运算符“::”。但通常情况下，在DESCRIPTION文件中列举要导入的包，这样在使用包时，相应要导入的包也会自动加载。

**3 Build**

在编辑好DESCRIPTION、hello.Rd、hello-package.Rd等文件后，点击Build按钮，如图18-9所示：



图18-9 Build的选择

Build & Reload 和 Check 选项提供了 R CMD的可视化操作，点选Check之后就相当于执行了Rcmd的 build 和 check 指令。执行结果将会显示于菜单下方窗口，如图18-10所示。

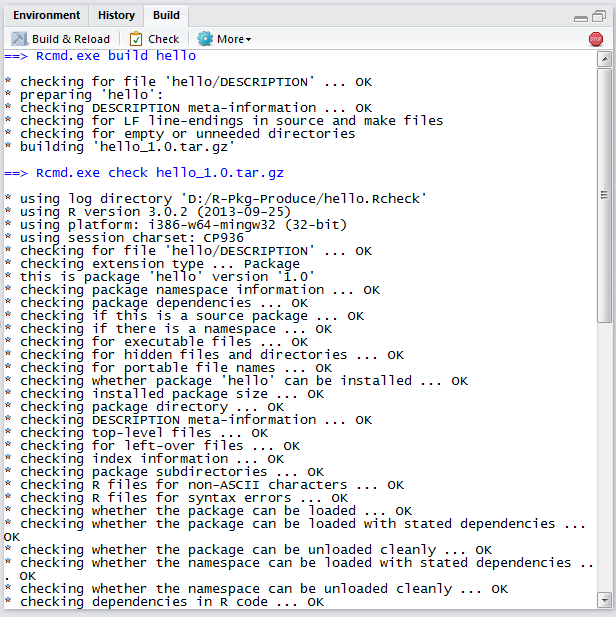


图18-10 完成创建过程

最后，我们会在先前创建工程时设定好的工作路径下找到被创建好的R包。将R包载入后执行得到如下结果：

|  |
| --- |
| >library(hello)  >hello(“world”)  [1] “Hello world” |