R 语言知识点汇编

Author: DavidSciMeow

2026-01-16

目录

留空

1 R 语言概述

1.1 R 的起源与发展

R 最早源自贝尔实验室的 S 语言的思想和实现，目标是为统计计算和图形展示提供一种交互式语言。1990 年代早期，Ross Ihaka 与 Robert Gentleman 在新西兰奥克兰大学基于 S 设计并实现了 R。随后 R 在全球范围内被学术界和工业界广泛采纳，形成了以 CRAN（Comprehensive R Archive Network）为核心的软件和文档分发体系。R 基金会（R Foundation）负责协调 R 的发展与发布，社区通过贡献包、文档和示例推动了生态的扩展。

1.1.0.1 演化 R 从最初作为统计研究工具演变为数据科学常用语言之一：核心语言保持稳定，扩展主要通过 CRAN、Bioconductor、GitHub 等渠道提供大量领域包（统计建模、可视化、机器学习、生物信息等）。近年出现了诸如 data.table、tidyverse、sparklyr 等重要项目，提升了 R 在大数据与工程化场景下的可用性。

1.1.0.2 R 的特点

- 开源免费，跨平台（Windows、macOS、Linux）。
- 面向向量和矩阵的计算模型，内置丰富的统计函数与绘图功能。
- 强大的包生态（CRAN、Bioconductor 等）和活跃社区支持。
- 交互式分析与脚本化工作流并重，便于探索性分析与可重复研究。
- 支持扩展的 C/Fortran 接口以及与数据库、大数据平台的集成能力。

1.2 安装与运行 R

1.2.0.1 获取 R 可从 CRAN 官网（https://cran.r-project.org）或各地镜像下载适用于不同操作系统的安装包；在部分 Linux 发行版上也可通过包管理器安装。

1.2.0.2 安装

- Windows：下载官方安装程序并按提示安装。
- macOS：下载 .pkg 安装包或使用 Homebrew（brew install r）。
- Linux：使用发行版的包管理器（如 apt、yum）或从源代码编译安装。

1.2.0.3 运行 R

- 交互式：R 命令进入交互式控制台；Windows 有 RGui。
- 脚本运行：使用 `Rscript your_script.R` 或 R CMD BATCH 在批处理/生产环境中执行脚本。
- IDE：常用 RStudio、VS Code（配合 R 扩展）提供更友好的开发环境。

1.3 安装与使用包

1.3.0.1 什么是包 包（package）是 R 的功能扩展单元，包含 R 代码、数据、文档、示例与测试。CRAN 是标准分发渠道，Bioconductor 面向生物信息学领域，GitHub 常用于开发版分发。

1.3.0.2 安装包

- 从 CRAN：install.packages(”pkgname”)。
- Bioconductor：使用 BiocManager::install(”pkg”)。
- GitHub：使用 devtools::install_github(”user/repo”) 或 remotes::install_github()。

1.3.0.3 载入与使用 使用 library(pkgname) 或 require(pkgname) 载入包；也可用 pkgname::function() 直接调用包内函数，避免污染全局命名空间。

1.3.0.4 卸载包 使用 remove.packages(”pkgname”) 从库中移除包。

1.3.0.5 包的命名空间 包拥有独立命名空间，导出的符号可被用户直接调用，未导出的符号仅供包内部使用。命名空间机制通过 NAMESPACE 文件声明导出与导入规则，:: 与 ::: 运算符用于访问导出和非导出对象（后者不推荐常用）。理解搜索路径（search()）与命名冲突有助于排查加载问题。

1.4 工作空间管理

1.4.0.1 工作空间概念 工作空间（workspace）指当前 R 会话中已创建的对象集合（变量、函数、数据框等）。常见操作包括列出对象、保存与恢复、清理及设置工作目录。

1.4.0.2 常用命令

- ls() 或 objects()：列出当前对象。
- rm(x)：删除对象，rm(list = ls()) 可清空当前会话对象。
- save(object, file = ”file.RData”)、load(”file.RData”)：持久化与恢复对象。
- save.image()：保存整个工作空间（生成 .RData）。
- getwd()、setwd()：查看与设置工作目录。
- 推荐使用 R 项目（RStudio 的 .Rproj）来管理文件与会话，提升可重现性。

1.5 集成开发环境 RStudio

1.5.0.1 什么是 IDE 集成开发环境（IDE）是将编辑器、控制台、调试器、项目管理、包与帮助浏览等功能整合在一起的工具，旨在提高开发效率与可视化操作体验。

1.5.0.2 RStudio 的主要功能与使用

- 脚本编辑与执行（逐行或选中运行）、交互控制台。
- 环境/变量窗格、历史记录、文件与包管理面板。
- 可视化输出、绘图浏览与导出功能。
- 调试工具（断点、step 等）、项目支持（.Rproj）、R Markdown 与交互式文档支持。
- RStudio Server 可在远程服务器上提供基于浏览器的 IDE。

1.6 帮助系统

R 提供多层次帮助：函数级别（?mean 或 help("mean")）、模糊搜索（??"regression" 或 help.search()）、包文档（help(package = "ggplot2")）、示例（example(fun)）和详细教程/长文档（vignette("vignette-name", package = "pkg")）。另外 help.start() 可启动本地 HTML 帮助索引。熟练使用帮助系统能显著加速学习与开发。

1.7 R 语言与大数据

1.7.0.1 与大数据平台的集成 R 与大数据生态有多种结合方式：

- 数据库连接：通过 DBI、RMySQL、RPostgres 等与关系型数据库直接交互。
- 分布式计算：sparklyr、SparkR 能将计算下推到 Apache Spark；RHadoop、rhdfs 等可与 Hadoop 生态集成。
- 内存与高性能包：data.table、dtplyr 在单机上处理大数据表现优异；bigmemory、ff 提供对超出内存数据的支持。
- 并行计算：parallel、foreach、future 等包支持多核与分布式任务并行化。

1.7.0.2 在数据科学中的角色 R 在数据清洗、统计建模、可视化与快速原型方面具有优势。尽管在极大规模（PB 级）数据处理上常与 Spark、数据库或 Python 互补，但 R 在建模实验、可视化展示与统计推断领域仍然是重要工具。

1.8 小结

本章概述了 R 的起源与演化、安装与运行、包管理、工作空间、RStudio IDE、帮助系统以及 R 与大数据的关系。后续章节将对数据类型、控制结构和函数等主题展开详细介绍。

2 数据类型与运算

2.1 基础知识

2.1.0.1 向量 向量是 R 中最基础的数据结构之一；一个向量包含同一类型的元素（numeric、integer、logical、character）。向量支持下标访问（从 1 开始），常见创建方式有 c(...)、seq()、rep()。

2.1.0.2 对象 在 R 中几乎一切都是对象：向量、矩阵、数据框、函数等都是对象，可以赋值给变量并传递给函数。

2.1.0.3 函数 函数是 R 的第一类对象，既有基本函数（如 sum(), mean()），也可自定义函数（function(x) { ... }）。函数参数支持位置与命名传递，可设默认值。

2.1.0.4 标识符与保留字 标识符用于命名对象，必须以字母或点开头（但不能以点后接数字），后续可包含字母、数字、点和下划线。R 有若干保留字（如 if, else, for, function, TRUE, FALSE, NA），不能用作变量名。

2.2 数据类型

类型	说明
数值型	numeric / double
整数型	integer（例如使用 1L 表示整数常量）
逻辑型	logical（TRUE / FALSE）
字符型	character（字符串，用双引号或单引号表示）
因子	factor（用于表示分类变量，底层用整数表示并带有标签）

2.2.0.1 基本数据类型

2.2.0.2 变量与常量 变量通过赋值创建（如 x <- 1 或 x = 1），R 没有语言层面的常量关键字，但可通过约定或包实现只读行为。

符号	含义
NA	缺失值（missing value）
NaN	不是数（not a number），通常为 0/0 的结果
Inf / -Inf	正/负无穷
NULL	表示空对象

2.2.0.3 特殊值

2.3 基本运算

2.3.0.1 运算符分类

2.3.0.2 索引与子集 R 支持多种索引方式：数字下标（从 1 开始）、逻辑下标、名称下标以及切片。使用 `[ ]`、`[[ ]]` 和 `$` 来访问不同层级或字段。

2.3.0.3 示例

优先级	类别	符号	说明
1	幂与一元运算	^	幂运算
1	幂与一元运算	+（一元）	一元正号
1	幂与一元运算	-（一元）	一元负号
2	乘除与模	*	乘法
2	乘除与模	/	除法
2	乘除与模	%%	取模（返回余数）
2	乘除与模	%/%	整数除法
3	加减	+	加法
3	加减	-	减法
4	关系运算	==	等于
4	关系运算	!=	不等于
4	关系运算	>	大于
4	关系运算	<	小于
4	关系运算	>=	大于等于
4	关系运算	<=	小于等于
5	序列与索引运算	:	序列生成运算符
6	元素级逻辑	&	元素级与
6	元素级逻辑	|	元素级或
7	短路逻辑	&&	短路与
7	短路逻辑	||	短路或
8	取反	!	逻辑取反
9	赋值	<-	左赋值
9	赋值	=	赋值（注意上下文差异）
9	赋值	->	右赋值
9	赋值	«-	全局赋值

数据结构	说明
向量（vector）	同质一维数组，最常用的数据结构。
矩阵（matrix）	同质二维数组，可用 matrix() 创建。
数据框（data.frame / tibble）	异质表格型结构，每列可为不同类型，常用于数据分析。
列表（list）	异质的一维容器，可包含任意对象（向量、数据框、函数等）。
因子（factor）	用于表示分类数据，便于统计建模。

操作	例子
创建向量	x <- c(1, 2, 3)
创建数据框	df <- data.frame(a = 1:3, b = c(”x”,”y”,”z”))
子集选择	df[df$a > 1,]、lst[[1]]、df$b

3 程序设计基础

3.1 控制流

3.1.1 顺序结构

程序按照书写顺序逐行执行，这是最基本的执行模型。程序由若干语句组成，语句按顺序依次计算并产生副作用或返回值。

3.1.2 分支结构

用于根据条件选择执行路径。常见形式有 ‘if‘、‘if-else‘、‘ifelse()‘ 以及 ‘switch()‘。示例：

```
if (x > 0) {
  print("positive")
} else if (x < 0) {
  print("negative")
} else {
  print("zero")
}
```

3.1.3 循环结构

用于重复执行代码。常见的循环有 ‘for‘、‘while‘ 和 ‘repeat‘。在 R 中优先考虑向量化操作或 ‘apply‘ 系列以提高性能。示例：

```
for (i in 1:5) {
  print(i)
}

i <- 1
while (i <= 5) {
  print(i)
  i <- i + 1
}
```

3.1.4 选择结构

‘switch()‘ 用于根据一个表达式的值选择多个分支，适合离散的选项分发。逻辑索引用于按条件筛选或赋值，是 R 风格的“选择”操作。

3.2 函数设计

3.2.1 声明、定义与调用

函数用 ‘function‘ 关键字声明，并通过名称调用：

```
f <- function(x, y = 1) {
  x + y
}
f(2)
```

3.2.2 返回值

函数的返回值为最后一个求值表达式的值，或显式使用 ‘return()‘ 返回。

3.2.3 函数的输入输出

函数参数支持位置参数、命名参数和默认值。建议为复杂函数写明参数含义并进行输入校验。

3.2.4 环境与范围

R 使用词法作用域（lexical scoping）。函数内部默认访问最近的环境；使用 ‘«-‘ 可修改外层环境变量，但应谨慎使用以避免副作用。

3.2.5 递归函数

递归函数在函数体内调用自身，适用于分治和递归定义的问题，但需注意终止条件与性能。示例：阶乘

```
fact <- function(n) {
  if (n <= 1) return(1)
  n * fact(n - 1)
}
```

3.3 编程规范与性能优化

3.3.1 使用脚本

把相关代码组织成脚本文件（‘.R‘），用 ‘source()‘ 加载；项目层面建议使用包结构或 ‘here‘/‘rproj‘ 管理路径。

3.3.2 编程规范

命名应有一致性（如驼峰或下划线）；写足够注释，避免“魔法数字”；把可复用逻辑封装为函数，添加文档与示例。

3.3.3 性能优化

优先使用向量化操作、预分配存储（如 ‘numeric(n)‘）、适当选择数据结构（‘data.table‘/‘tibble‘），并用 ‘Rprof()‘、‘profvis‘ 等工具定位瓶颈。示例建议：

```
# 不推荐：在循环中逐步扩展向量
v <- c()
for (i in 1:10000) v <- c(v, i)

# 推荐：预分配
v <- integer(10000)
for (i in 1:10000) v[i] <- i
```

更多进阶优化包括并行计算（‘parallel‘, ‘future‘）、使用 C/C++ 接口（‘Rcpp‘）等。

4 类与对象

4.1 面向对象程序设计方法

4.1.1 结构化程序设计方法的问题

结构化程序设计强调过程和步骤，但在处理复杂系统时容易导致代码耦合、全局状态增多、可维护性差和扩展困难。

4.1.2 对象与类的概念

对象是包含状态（数据）和行为（方法）的实体；类是描述对象结构和行为的模板。通过类可以封装数据和操作，隐藏实现细节。

4.1.3 面向对象程序设计的特点

- 封装：将数据和操作封装在对象中。- 继承：通过基类扩展新类复用代码。- 多态：同一接口在不同类中有不同实现。

4.1.4 R 中类的体系

R 支持多种 OOP 系统：非正式的 S3，形式化的 S4，以及引用语义的 Reference Class（和 R6 包提供的现代引用类）。不同系统在类定义、方法分派和数据访问语义上有所区别。

4.2 S3 类

4.2.1 S3 类定义

S3 是一种轻量的面向对象机制，基于在对象上设置 ‘class‘ 属性。无需事先声明类。

4.2.2 创建 S3 类对象

示例：

```
person <- list(name = "Alice", age = 30)
class(person) <- "person"
person
```

4.2.3 S3 类的泛型函数

S3 使用基于 ‘UseMethod()‘ 的泛型函数分派。例如 ‘print()‘、‘summary()‘ 是泛型，按对象的 ‘class‘ 调用对应方法。

4.2.4 定义 S3 类的方法

通过命名约定 ‘generic.class‘ 定义方法：

```
print.person <- function(x, ...) {
 cat("Person:", x$name, "(age", x$age, ")\n")
}
```

4.2.5 编写 S3 类的泛型函数

自定义泛型：

```
greet <- function(x, ...) UseMethod("greet")
greet.person <- function(x, ...) cat("Hello,", x$name, "!\n")
```

4.3 S4 类

4.3.1 S4 类的定义

S4 是更严格的 OOP 系统，要求在创建类前显式声明类和槽（slots），并支持形式化的类检查。

4.3.2 创建 S4 类的对象

定义与实例化示例：

```
setClass("Person",
     slots = list(name = "character", age = "numeric"))
p <- new("Person", name = "Bob", age = 40)
```

4.3.3 访问插槽

使用 ‘@‘ 操作符访问 S4 对象的插槽：

```
p@name
p@age <- 41
```

4.3.4 S4 类的泛型函数

S4 使用 ‘setGeneric()‘ 和 ‘setMethod()‘ 定义泛型和方法，更适合较大项目和包开发。示例：

```
setGeneric("greet", function(x) standardGeneric("greet"))
setMethod("greet", "Person", function(x) cat("Hi,", x@name, "\n"))
```

4.3.5 定义 S4 类的方法

使用 ‘setMethod()‘ 为特定类签名添加实现，支持多参数的签名匹配。

4.4 引用类

4.4.1 定义引用类

引用类（Reference Classes）提供引用语义（mutable objects），适用于需要在多个地方共享可变状态的场景。可通过 ‘setRefClass()‘ 定义。

4.4.2 创建引用类的对象

示例：

```
Counter <- setRefClass("Counter",
 fields = list(count = "numeric"),
 methods = list(
  increment = function() count <<- count + 1,
  get = function() count
 ))
c <- Counter(count = 0)
c$increment()
c$get()
```

4.4.3 访问和修改引用对象的域

通过 ‘$‘ 访问字段与方法，修改字段会改变对象本身（引用语义）。

4.4.4 引用类的方法

方法在类定义中直接声明，使用 ‘self‘ 或直接引用字段来操作状态。

4.5 继承

4.5.1 S3 类的继承

S3 通过在 ‘class‘ 属性中指定多个类名实现继承与方法回退，例如 ‘c(”child”, ”parent”)‘，泛型会首先查找 ‘child‘ 方法，再回退到 ‘parent‘。

4.5.2 S4 类的继承

S4 使用 ‘contains‘ 在 ‘setClass()‘ 中声明继承关系：

```
setClass("Employee", contains = "Person",
     slots = list(empId = "character"))
```

4.5.3 引用类的继承

Reference Classes 支持通过 ‘contains‘ 参数继承其他引用类。

4.5.4 多重继承

S4 支持多重继承（多个 ‘contains‘），需要注意潜在的方法冲突与槽名冲突，通常通过明确方法签名解决歧义。

5 数据结构与数据处理

5.1 向量

5.1.1 创建向量

使用 c()、seq()、rep() 等创建向量：

```
x <- c(1, 2, 3)
y <- seq(1, 10, by = 2)
z <- rep(0, times = 5)
```

5.1.2 使用索引访问向量

使用下标访问（从 1 开始）：`x[1]`；逻辑索引和名称索引也可用。

5.1.3 循环补齐

循环中逐步增长向量效率低，推荐预分配或使用向量化操作。

5.1.4 向量比较

对向量进行比较会返回逻辑向量，例如 ‘x > 0‘。

5.1.5 按条件提取元素

使用逻辑索引提取满足条件的元素：`x[x > 0]`。

5.2 矩阵和数组

5.2.1 创建矩阵

用 matrix() 或 rbind/cbind 创建矩阵：

```
m <- matrix(1:6, nrow = 2)
```

5.2.2 线性运算

支持矩阵乘法（‘

5.2.3 使用矩阵索引

通过 ‘[i, j]‘ 访问元素，行/列切片返回矩阵或向量。

5.2.4 Apply 函数族

使用 ‘apply()‘、‘rowSums()‘、‘colMeans()‘ 等对矩阵按维度操作：

```
apply(m, 1, sum) # 每行求和
```

5.2.5 多维数组

使用 ‘array()‘ 创建多维数组，使用 ‘[i,j,k]‘ 索引访问。

5.3 数据框

5.3.1 创建数据框

使用 ‘data.frame()‘ 或 ‘tibble::tibble()‘ 创建数据框：

```
df <- data.frame(a = 1:3, b = c("x","y","z"))
```

5.3.2 访问元素

使用 ‘df$col‘、‘df[i, j]‘、‘df[”col”]‘ 访问列和单元格；使用 ‘subset()‘ 或 ‘dplyr‘ 进行筛选。

5.3.3 使用 SQL 语句查询数据

可用 ‘sqldf‘ 包在 R 中对数据框执行 SQL 查询：

```
library(sqldf)
sqldf("select * from df where a > 1")
```

5.4 因子

因子用于表示分类数据，常用于统计建模与绘图；使用 ‘factor()‘ 创建并设置水平（levels）与标签（labels）。

5.5 列表

列表是异质容器，可存放不同类型对象：

```
lst <- list(name = "Alice", scores = c(90, 85))
```

使用 ‘[[ ]]‘ 或 ‘$‘ 访问元素。

5.6 数据导入与导出

5.6.1 数据文件的读写

常用函数：`read.csv/read.table`、`write.csv`；现代包：`readr::read_csv`、`readr::write_csv`。

5.6.2 rio 包

‘rio‘ 提供简洁的 ‘import()‘/‘export()‘ 接口，自动识别文件格式，方便常见数据交换。

5.6.3 数据编辑器

可使用 ‘edit()‘、‘utils::View()‘ 或 RStudio 的数据查看器进行交互式编辑/查看。

5.7 数据清洗

5.7.1 数据排序

使用 ‘order()‘、‘dplyr::arrange()‘ 对数据排序。

5.7.2 数据清洗的一般方法

- 缺失值处理（删除、填补、插值）- 重编码与类型转换（如将字符转因子）- 重命名列、去重、格式化日期

5.7.3 mice 包

‘mice‘ 提供多重插补方法处理缺失数据，适合较复杂的缺失模式分析。

6 绘图和数据可视化

6.1 基本图形和绘图函数

6.1.1 基础图形的创建

使用基础绘图函数 ‘plot()‘, ‘barplot()‘, ‘hist()‘, ‘boxplot()‘ 等创建常见图形。例如：

```
x <- rnorm(100)
plot(x, main = "散点图示例", xlab = "索引", ylab = "值")
hist(x, breaks = 20, main = "直方图")
```

6.1.2 新增绘图窗口

在交互式会话中用 ‘dev.new()‘（或 RStudio 的绘图窗格）打开新绘图设备；也可指定设备类型如 ‘png()‘、‘pdf()‘。

6.1.3 导出图形

使用 ‘png()‘, ‘jpeg()‘, ‘pdf()‘, ‘svg()‘ 等函数将绘图输出到文件：

```
png("figure.png", width = 800, height = 600)
plot(x)
dev.off()
```

6.2 调整绘图参数

6.2.1 自定义特征

通过 ‘par()‘ 设置全局绘图参数（如边距 ‘mar‘、布局 ‘mfrow‘ 等）。

6.2.2 调整符号与线条

使用参数 ‘pch‘（点型）、‘cex‘（大小）、‘lty‘（线型）、‘lwd‘（线宽）自定义样式：

```
plot(x, pch = 19, cex = 0.6)
lines(smooth.spline(x), lty = 2, lwd = 2)
```

6.2.3 调整颜色

使用 ‘col‘ 指定颜色，可用颜色名、十六进制或 ‘rainbow()‘、‘heat.colors()‘ 等调色函数。

6.2.4 调整标签和标题文本

使用 ‘main‘、‘xlab‘、‘ylab‘、‘cex.main‘、‘cex.lab‘ 控制标题与标签文本大小与内容；用 ‘mtext()‘ 添加边缘文本。

6.3 其他自定义元素

6.3.1 坐标轴

使用 ‘axis()‘ 自定义刻度与标签，关闭自动坐标 ‘axes = FALSE‘ 后手动绘制。

6.3.2 次要刻度线

基础绘图没有内建次刻度，可用 ‘axis()‘ 与 ‘abline()‘ 手动绘制细分刻度线或借助 ‘Hmisc::minor.tick()‘。

6.3.3 网格线

使用 ‘grid()‘ 或 ‘abline()‘ 添加网格线：

```
plot(x)
grid(nx = NULL, ny = NULL)
```

6.3.4 叠加绘图

通过在已有图上使用 ‘points()‘, ‘lines()‘, ‘text()‘ 等函数叠加元素；要保持尺寸比例可使用 ‘par(new = TRUE)‘ 小心覆盖坐标系。

6.3.5 图例

使用 ‘legend()‘ 添加图例，指定位置、符号与颜色：

```
legend("topright", legend = c("数据1","拟合"), pch = c(1, NA), lty = c(NA,2))
```

6.3.6 标注

使用 ‘text()‘、‘arrows()‘、‘points()‘ 等添加标注与箭头以突出要点。

6.4 描述性统计图

6.4.1 柱状图

使用 ‘barplot()‘ 绘制分组或堆叠柱状图。

6.4.2 饼图

使用 ‘pie()‘ 绘制饼图（注意：饼图在展示百分比时应谨慎使用）。

6.4.3 直方图

使用 ‘hist()‘ 控制 ‘breaks‘ 与 ‘freq/density‘ 参数显示频数或密度。

6.4.4 箱型图

使用 ‘boxplot()‘ 比较分组分布并检测异常值。

6.4.5 三维绘图

可用基础函数 ‘persp()‘ 绘制三维表面，或使用 ‘scatterplot3d‘/‘rgl‘ 包做交互式三维散点图。

6.5 动态图形

6.5.1 保存 GIF

使用 ‘magick‘ 或 ‘animation‘ 包将多帧图像合成为 GIF，或使用 ‘gifski‘ + ‘png‘ 帧序列。

6.5.2 gganimate 包

`gganimate` 基于 `ggplot2` 可创建时间序列动画，典型流程：用 `ggplot()` 创建静态图，加入 `transition_*()` 定义帧过渡，最后用 `animate()` 或 `anim_save()` 导出。

7 总结

7.1 学习建议

- 从基础数据结构和向量化思想入手
- 熟练掌握 `dplyr` 与 `ggplot2` 以提高数据处理效率
- 学会使用包管理与项目化工作流保证可重复性

7.2 扩展阅读

推荐书目与在线资源：CRAN、RStudio 教程、Hadley Wickham 的书籍与资料。