从动态作用域到词法作用域：SAS 与 R 作用域规则的比较研究

Chen Ling[1] 和 Yachen Wang[1]艾伯维公司,
美国伊利诺伊州莱克县 通讯作者：Chen Ling
(ling0152@umn.edu)

摘要

变量作用域决定变量在编程语言中的可访问方式和位置，对代码效率和组织至关重要。本文考察 SAS 和 R 各自不同的作用域规则，重点关注 SAS 的动态作用域和 R 的词法作用域。在 SAS 中，动态作用域利用符号表，在运行时通过动态搜索活动的宏层来解析变量。与之相对，R 采用词法作用域，使用环境根据函数定义时所处的结构来解析变量。

示例阐明了这些作用域策略之间的差异，并展示了它们对代码行为的影响。此外，本文概述了检查 SAS 符号表和 R 环境中变量的方法，为调试和优化提供实用见解。文中还讨论了控制两种语言中变量作用域的策略，以提升代码的精确性和可靠性。本研究帮助程序员深入理解如何优化变量管理，从而改进其 SAS 和 R 编程实践。

1. 引言

变量作用域是一个关键概念，它影响变量在代码库中的访问方式和位置。这一概念在函数式编程中尤为重要，因为 SAS 中的宏和 R 中的函数等构造都高度依赖明确定义的作用域规则，以确保准确性和效率。由于不同编程语言提供了不同的作用域机制，理解这些差异对于希望编写精准、可靠代码的开发者至关重要。

尽管作用域规则十分重要，但关于 SAS 和 R 这两种在制药行业广泛使用、且采用不同作用域方法的语言之间作用域规则差异的讨论却寥寥无几。SAS 采用动态作用域，依靠符号表在运行时管理变量解析。这种方法会逐层搜索当前活动的宏调用，以查找变量，从而为变量管理提供灵活性。另一方面，R 使用词法作用域（Wickham，2019），由环境根据函数定义的位置来确定变量解析方式。这种方法使变量行为更具结构稳定性和可预测性。

随着许多组织从 SAS 转向 R（Ling & Wang, 2025），了解这些作用域规则的差异对于确保编程结果准确至关重要。我们通过详细示例探讨动态作用域与词法作用域的影响，阐明它们对代码行为的作用。此外，本文还讨论了在这两种语言中检查和控制变量作用域的策略。本文旨在揭示不同的作用域规则如何影响程序结果，从而帮助从 SAS 转向 R 的程序员更顺利地完成过渡。

2. 什么是变量作用域

变量作用域是函数式编程中的一个基本概念，它决定了变量在代码库中的声明、修改和访问位置。它定义了支配变量可见范围和生命周期的规则，而这些规则会显著影响程序的设计和行为。理解作用域对于编写准确、高效的代码至关重要，因为作用域决定了函数和宏如何引用其内部使用但未定义的变量。

在 SAS 和 R 中，通常有两种作用域：全局作用域和局部作用域。如下图1所示，全局变量定义在全局作用域中，而在宏或函数内部定义的变量则处于局部作用域。全局变量在整个 SAS/R 会话中始终可访问，而局部变量仅限于各自的作用域，并会在作用域结束后被丢弃。在下面的示例中，要计算局部作用域中 z 的值，我们需要 x 和 y 的值。由于变量 x 未在局部作用域中定义

因此，在 SAS 和 R 中都会从全局作用域获取该变量。不过，两种语言中这种“获取”机制（作用域规则）有所不同，本文将对此作进一步讨论。

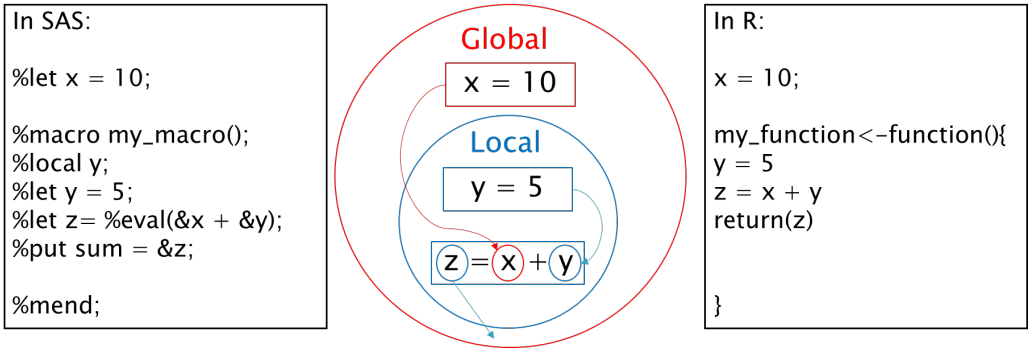

图1. SAS 和 R 中变量作用域的示例

3. 变量作用域由什么驱动

在 SAS 中，变量作用域依赖一种称为符号表的特殊内存区域（Buchecker, 2020）。符号表用于管理宏变量，对动态代码替换至关重要。符号表主要分为两类：全局符号表和局部符号表，分别对应前文提到的全局作用域和局部作用域。SAS 会话启动时会创建唯一的全局符号表，并在 SAS 会话结束时将其删除。不过，在特定条件下，宏执行期间会临时创建各种局部符号表，并在执行结束后将其删除。

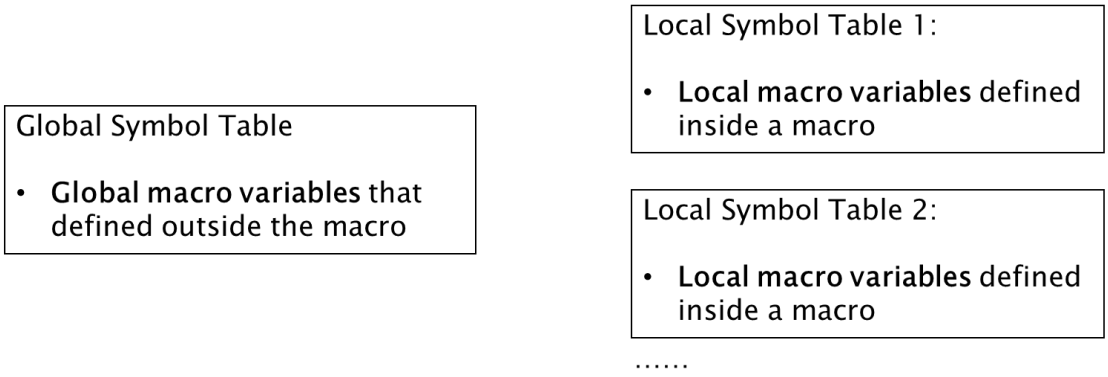

图2. SAS 中符号表的演示

与 SAS 的符号表类似，R 使用一种称为环境的内存区域来管理变量及其值（Ling & Wang, 2025）。R 不使用全局和局部符号表，而是通过全局环境和局部环境来实现全局作用域和局部作用域。在下方的图3中，左侧是源代码，右侧则展示了代码执行过程。

- 第一步，R 在全局环境中创建两个对象：将值 6 赋给 y，并将一个函数赋给 h（用一个小粉色符号表示）。
- 第二步，通过 h(1) 调用函数 h()，从而为 h() 启动一个执行环境。在此环境中，变量 a 被设为 2，而 x 默认为 1。
- 第三步，函数计算 x 和 a 的和，并将该结果返回，以便将其赋值给全局环境中的 z。函数 h() 完成执行并返回值后，执行环境将被丢弃。

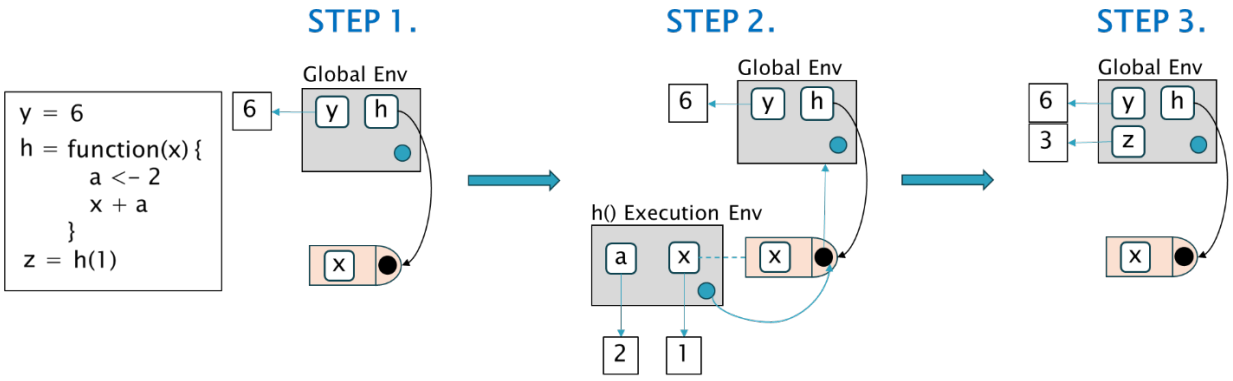

图 3. R 中环境的示意图

大多数环境都有一个更高层级的环境，称为父环境；即使全局环境也有一个父环境：空环境。值得注意的是，空环境是唯一没有父环境的环境（Wickham，2019）。理解某个环境的父环境至关重要，因为它决定了该环境将在何处查找自由变量的值。自由变量是指在函数内部使用、既不是形式参数，也不是在函数体内定义的局部变量。函数的执行环境的父环境取决于函数的定义位置。例如，在图 3 中，h() 执行环境的父环境是全局环境，因为函数 h() 定义在全局环境中。

4. SAS 和 R 中的作用域规则

4.1 SAS 中的动态作用域

在 SAS 中，宏变量的作用域取决于它们是在全局上下文还是局部上下文中定义的。默认情况下，在宏定义之外创建的任何宏变量都会自动存放在全局符号表中。在宏定义内部，作用域规则会优先查找现有的局部变量，这意味着新定义的宏变量是宏内的局部变量。如果宏有参数，这些变量会自动成为该宏的局部变量。SAS 会先在局部符号表中查找宏变量；如果没有找到，则会沿着调用宏逐层向上查找，最后才检查全局符号表。

现在，你可能已经了解 SAS 中作用域的工作方式了。你能否帮我预测运行 SAS 程序 5 中的所有代码后，x1 和 y1 的值?

```
/*动态作用域示例*/
%let x1=9;
%let y1=7;
%macro 我的_宏2;
%let x1=19;
%let y1=17;
%mend;

%我的_宏2()
```

SAS 程序1 动态作用域示例1　　ing

这个测试的答案是：x1=19 和 y1=17。有些人可能会发现答案令人惊讶：为什么 SAS 会使用在宏内部赋值的“局部”变量来替换全局变量？或者我们也可以这样问：SAS 宏执行时，在宏内部定义的所有宏变量都是局部变量吗？答案显然是否定的，并非总是如此。在这种情况下，当我们在 my_宏2 中将 19 赋给 x1 时，SAS 会先在局部符号表中查找 x1。如果在局部符号表中找不到 x1，SAS 就会继续在全局符号表中查找。由于 x1 已经存在于全局符号表中，因此在这种情况下，SAS 会直接将全局符号表中的 x1 值替换为 19。

让我们深入探究，了解幕后究竟发生了什么。与大多数采用**词法作用域**（也称为静态作用域）的现代编程语言不同，SAS 通常采用**动态作用域**作为其作用域规则（Ling & Wang, 2025）。一方面，词法作用域会根据变量或函数被调用处的词法

（文本）内容来查找变量的值。另一方面，在动态作用域中，如果变量的作用域是某个宏，那么其作用域就是该宏执行期间。这意味着变量赋值的判定取决于运行时上下文，因此难以预测。由于动态作用域是在编译完成后的执行期间才起作用，宏处理器和编译器都无法判断某条赋值语句究竟是要赋值给局部宏变量，还是赋值给运行时可能存在的更高层级宏变量（Joe, 2016）。

来看一个更直观的动态作用域示例，如图4所示。左侧是展示嵌套宏结构中动态作用域的示例代码。宏 %inner 和 %outer 在全局作用域中定义，其中 %inner 在 %outer 内部调用。为了说明这一过程，我们用一张图来展示其背后的运行机制。这里定义了两个 x：一个在全局作用域中，另一个在 %outer 的局部作用域中，它们分别存储在全局符号表和 %outer 的局部符号表中。当 SAS 在 %inner 中解析 &x 的值时，会发现 %inner() 内没有对 x 的局部定义。因此，它会在运行时所在的作用域中查找 x。由于 %inner 在 %outer 内部调用，也就是说宏 inner 在 outer 中运行，SAS 会将 outer 中定义的 x=6 作为写入日志的值。

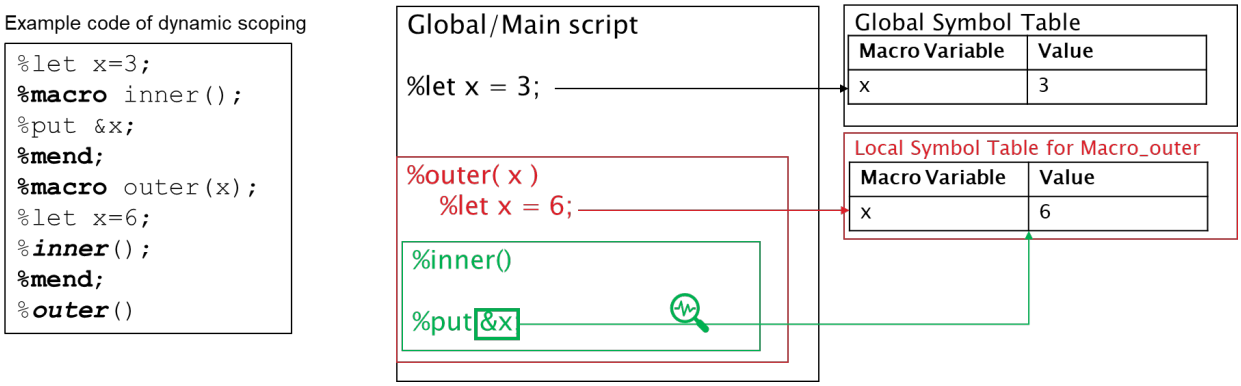

图4. SAS 中动态作用域的示例

4.2 R 中的词法作用域

要理解 R 的作用域规则，我们必须了解 R 如何为自由变量绑定值。如果在函数内部找不到某个变量的值，那么 R 会在定义该函数的环境中查找（Peng，2022）。如果在此环境中搜索失败，R 将继续在父环境中搜索。该过程会沿着连续的父环境进行，直到到达顶层环境；顶层环境通常是全局环境（工作区）或某个程序包的命名空间。超出顶层环境后，搜索会继续沿着搜索列表进行，直到遇到空环境。

接下来，让我们通过一个示例来加深理解，如图5所示。在函数 f3() 的执行环境中，变量 y 被设为 x*2，但 x 不在其环境中。为了找到 x 的值，搜索会从 f3() 的父环境 f2() 开始。由于在 f2() 的环境中找不到 x，搜索将沿着“父链”继续向上，直到顶层。幸运的是，我们在全局环境中找到了 x，因此 y 将被赋值为 4（2*2）。

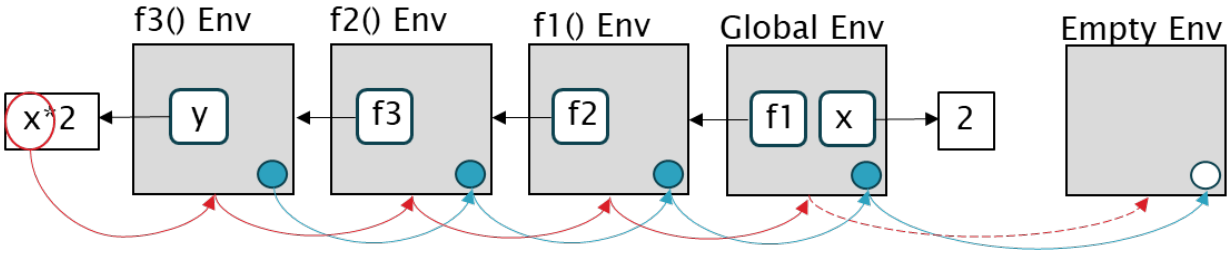

图5. 词法作用域示例 i n R

与大多数现代编程语言一样，R 使用词法作用域。词法作用域不考虑函数何时被调用，只取决于函数定义的位置。让我们回顾一下图4中的示例；下面是图6。左侧是结构相似但结果不同的示例代码。右侧

展示了代码的图示。由于 outer() 和 inner() 都是在全局环境中定义的，全局环境就是它们各自的父环境。当在 outer() 内部调用 inner() 时，它首先会在自己的局部环境中查找 x。由于此处没有定义 x，它便继续在 inner() 的父环境，也就是全局环境中查找，而不是像在 SAS 中那样在 outer() 的环境中查找。最后，它会找到全局环境中定义为 3 的 x，并将其打印出来。

Example code of lexical scoping

```
x=3
inner<-function(){
  print(x)
}
outer<-function(){
  x=6
  inner()
}
outer()
```

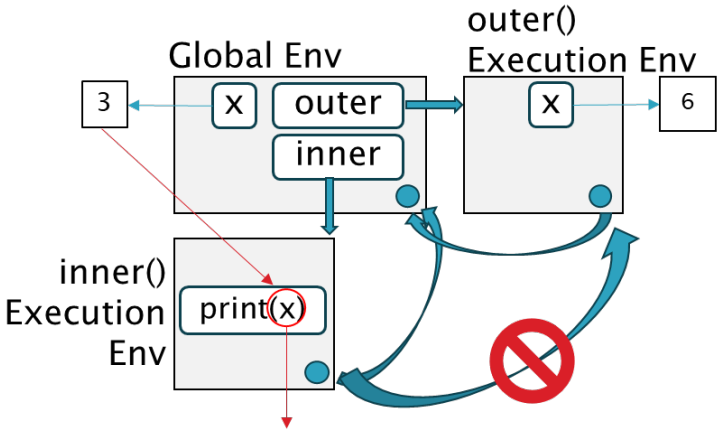

图6. R 中词法作用域的示例

5. 检查作用域中的变量

5.1 检查 SAS 中的符号表

调试并验证宏变量的放置位置，对于确保 SAS 程序的准确性至关重要。带有 _user_ 关键字的 %PUT 语句可用于将所有用户定义的宏变量、其值及各自的作用域列表写入 SAS 日志。在 SAS 程序 2 中，%put _user_ 用于在日志中打印全局符号表和局部符号表。如图1所示，全局符号表中存在一些额外变量；请注意，这些是 SAS 系统生成的变量，大多数情况下可以忽略。

```
/*检查符号表的示例*/
%let y=8;
%let z=9;
%macro lazy(x=1,y=%eval(&x*10),z=%eval(&a+&b));
%let x=2;
%let a=3;
%let b=4;
%put (&x &y &z);
%put _user_;
proc print data=sashelp.vmacro;
run;
%mend;

%惰性()
```

SAS 程序 2 符号表检查示例

```
(2 20 7)
LAZY A 3
LAZY B 4
LAZY X 2
LAZY Y %eval(&x*10)
LAZY Z %eval(&a+&b)
GLOBAL SASWORKLOCATION "/op
GLOBAL SYSSTREAMINGLOG true
GLOBAL Y 8
GLOBAL Z 9
```

图 7 日志中打印的局部和全局符号表

或者，也可以访问 sashelp.vmacro 视图。该视图提供包含所有宏变量及其名称、值和作用域的完整数据集，使用户能够直接有效地检查和管理局部符号表和全局符号表。

Obs	scope	name	offset	value
1	LAZY	A	0	3
2	LAZY	B	0	4
3	LAZY	X	0	2
4	LAZY	Y	0	%eval(&x*10)
5	LAZY	Z	0	%eval(&a+&b)
6	GLOBAL	SASWORKLOCATION	0	"/opt/SASWORK/F
7	GLOBAL	SYSSTREAMINGLOG	0	true
8	GLOBAL	Y	0	8
9	GLOBAL	Z	0	9

图 8 sashelp.vmacro 的内容

5.2 检查环境中的对象

在 R 中，我们可以使用 ls() 或 objects() 返回一个字符向量，其中包含环境中对象的名称。若要同时获取对象名称及其对应的值，可以使用 ls.str() 函数。对于变量，该函数会返回对象的类别及其值；对于数据集，则会返回其结构。

检查全局环境中对象的另一种方法是查看 RStudio 中的环境选项卡（图 9）。在此选项卡中，你可以找到全局环境中的所有对象。

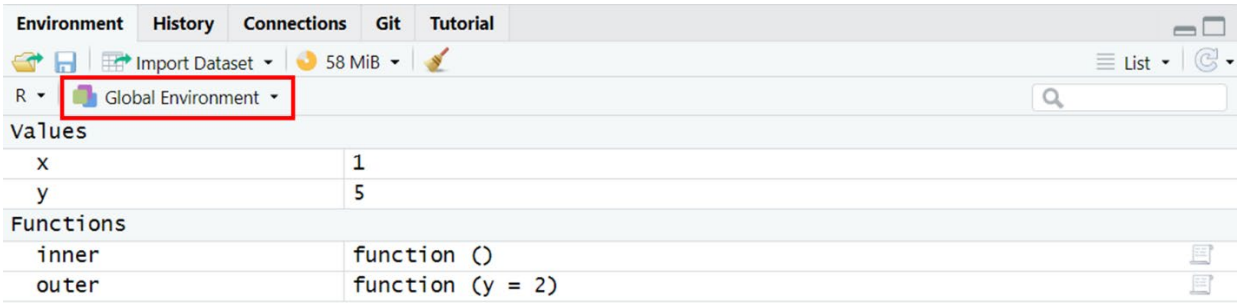

图 9. RStudio 中的环境选项卡

6. 控制变量作用域

6.1 在 SAS 中控制变量作用域

如前所述，SAS 宏通常采用动态作用域，不过，也可以显式指定宏变量为局部变量，从而实现词法作用域。在 SAS 程序 3 中，x2 被用作宏的参数，因此会自动成为局部变量。SAS 还提供了 %local 语句，可将 y2 设为局部宏变量。由于 x2 和 y2 都是局部变量，全局 x2 和 y2 的值不会改变。

%local 语句可确保宏变量限定在声明它的宏的局部环境中。如果该宏变量已存在，%local 将被忽略；否则，它会在局部符号表中创建一个宏变量。良好的做法是，始终使用 %local 将宏内部定义的变量指定为局部变量，以避免冲突。

此外，我们可以使用 %global 明确指定宏中的变量为全局变量。不过，对于多层嵌套作用域结构，无法强制将变量置于全局作用域与最局部作用域之间的中间作用域。

```
/*显式指定局部宏变量的示例*/ %let x2=9; %let y2=7;
%macro my_macro2(x2); %local y2;
```

```
%let x2=19;
%let y2=17;
%mend; %我
的_宏2()
```

SAS 程序 3 显式指定局部宏变量的示例

6.2 在 R 中控制变量作用域

在 R 中，<- 运算符用于为当前环境中的变量赋值。如果需要在父环境中赋值，则使用超赋值运算符 <<-。或者，也可以通过使用 .GlobalEnv$variable <- value 为全局环境中的变量赋值来控制作用域，这与 SAS 中的 %global 类似。

然而，通常不建议使用这些方法，因为它们可能会更改预期作用域之外的变量，从而产生副作用，影响范围超出函数显式返回的内容。这可能导致意外问题，尤其是对于不熟悉相关函数的用户而言。例如，意外的结果可能会覆盖调用环境中已有的变量，进而影响后续代码。建议在编程中避免使用这些作用域控制技术，以保持清晰并防止意料之外的后果。

7. 结论

总而言之，变量作用域是函数式编程中至关重要的基础要素，会显著影响 SAS 和 R 中变量的访问与管理方式。本文比较了 SAS 的动态作用域（通过符号表提供运行时灵活性）与 R 的词法作用域（通过环境提供稳定性和可预测性）。通过示例代码，本文阐明了这些作用域规则上的差异如何导致代码执行产生不同结果，并加深了对其实际影响的理解。检查和控制变量作用域的实用方法，也进一步帮助程序员提高代码的精确性和可靠性。对于在这两种语言之间转换的开发者而言，这种理解尤其有价值，因为它能帮助他们有效优化编程实践。归根结底，深入理解这些作用域规则有助于获得更准确的编程结果，并在同时使用 SAS 和 R 的环境中实现更顺畅的转换。

参考文献

Buchecker, M.（2020）。《全球思考，局部行动：理解全局和局部宏符号表》。*SAS* 全球论坛 *2020*。线上。取自 https://support.sas.com/resources/papers/proceedings20/4160-2020.pdf Joe。（2016年2月22日）。为什么 *SAS* 宏变量默认不是局部作用域？取自 Stack Overflow：https://stackoverflow.com/questions/35533276/why-arent-sas-macro-variables-local-scope-by-default Ling, C. 和 Wang, Y.（2025）。《惰性求值：SAS 宏与 R 函数的比较分析》。*SESUG 2025* 会议论文集。北卡罗来纳州卡里：美国东南部 SAS 用户组。Ling, C. 和 Wang, Y.（2025）。TLFQC：一个基于 R Shiny 的高兼容性平台，用于自动化、无代码生成和验证 TLF。*PharmaSUG 2025* 会议论文集。圣迭戈。取自：https://pharmasug.org/proceedings/2025/OS/PharmaSUG-2025-OS-076.pdf Ling, C. 和 Wang, Y.（2025）。《在 R 中编写 SAS 宏？R 函数可以帮忙！》 *PharmaSUG 2025* 会议论文集。加利福尼亚州圣迭戈。取自：https://pharmasug.org/proceedings/2025/AP/PharmaSUG-2025-AP-078.pdf Peng, R. D.（2022）。《数据科学的 *R* 编程》。Wickham

’ H. (2019). 《*Advanced R*》（第2版）。取自 https://adv -r.hadley.nz/index.html