惰性求值：SAS 宏与 R 函数的比较分析

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

摘要

惰性求值是一种强大的技术，它可以通过将求值推迟到需要结果时再执行来优化代码执行，从而提高效率。在大多数现代编程语言（如 R）中，惰性求值通常应用于函数参数。然而，SAS 中惰性求值的应用尚未得到广泛探讨。本文聚焦于 SAS 宏和 R 函数中的惰性求值机制，并对驱动这些过程的底层原理进行比较分析。

R 的惰性求值由一种名为 Promise 的数据结构驱动，它会推迟求值，并且在需要值之前不会占用内存，采用按需调用策略。另一方面，SAS 通过其符号表实现惰性求值，使用内存存储参数，并按名称调用。这些惰性求值策略的差异会显著影响 R 函数和 SAS 宏的结果。本文通过考察这些不同的方法，结合示例阐明惰性求值对编程效率的影响。随着从 SAS 转向 R 在制药行业日益普遍，了解这些技术有助于程序员优化代码，提高效能。本文的探讨可作为提升这两种语言编程能力和性能的指南。

1. 引言

在数据分析和统计编程中，提高效率并降低计算开销至关重要。提升效率的一种方法是采用惰性求值——这是一种将表达式的求值延迟到实际需要其结果时才进行的技术。作为现代编程语言之一，R 对函数参数采用惰性求值，利用一种称为 Promise 的数据结构和按需调用求值策略，将计算推迟到必要时才执行。这种方法能够提高效率，减少不必要的资源消耗，并使流程更加精简。尽管 R 的惰性求值实现广为人知，但 SAS 宏中的应用仍鲜有探讨。

几十年来，SAS 一直是制药行业编程工具包中的传统选择，以强大的数据分析能力和宏功能而闻名。虽然 SAS 使用符号表存储变量，这本身支持延迟计算，但它采用的是按名调用策略，这与 R 的按需调用方法有着根本区别。

了解这些不同的机制至关重要，尤其是许多制药公司正从 SAS 转向 R（Ling & Wang, 2025）。本文旨在阐明 SAS 宏与 R 函数在惰性求值方面的差异，并提供见解和示例，帮助程序员优化代码。深入探究这些细微差异，程序员便能发挥各语言的优势，提升编程效率。

2. SAS 和 R 将参数存储在哪里

要理解 SAS 和 R 等编程语言中的惰性求值，一个基本方面是考察参数存储在何处以及如何存储。这种存储机制会直接影响表达式的求值方式和时机，进而影响代码执行的效率和行为。

2.1 R 中的环境

R 使用一个称为环境的内存区域来管理变量及其值。环境主要分为两种：全局环境和局部环境，分别对应全局和局部作用域。下方左侧是源代码，右侧展示代码执行过程（Ling & Wang, 2025）。

- 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（以一个小小的粉色符号表示）。

- 第二步中，通过 h(1) 调用函数 h()，从而为 h() 创建一个执行环境。在此环境中，变量 a 被设为 2，而 x 的默认值为 1。
- 第三步中，函数计算 x 与 a 的和，并返回结果，将其赋给全局环境中的 z。函数 h() 执行完毕并返回值后，其执行环境便会被丢弃。

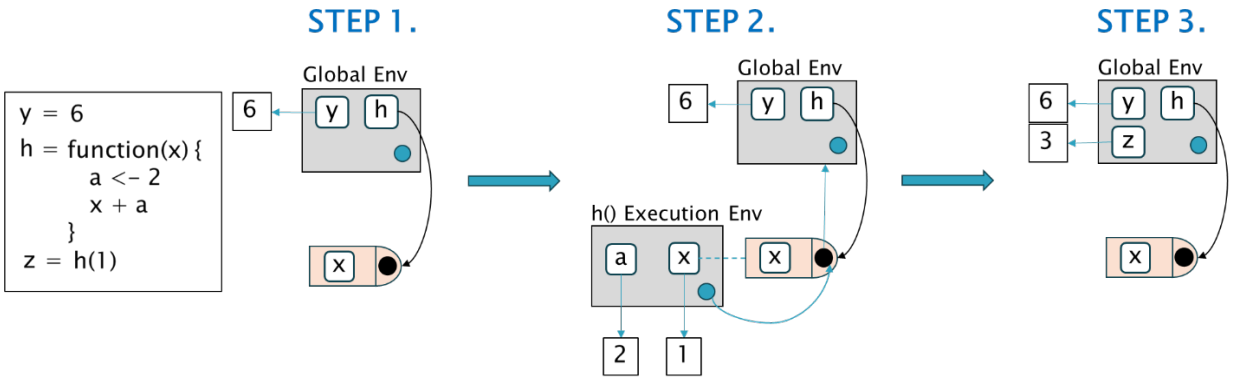

图 1. R 中环境的演示

2.2 SAS 中的符号表

在 SAS 中，变量作用域依赖于一个称为符号表的特殊内存区域，它负责管理宏变量，对动态代码替换至关重要。与 R 中的全局和局部环境不同，SAS 使用全局和局部符号表来管理全局和局部作用域。SAS 会话启动时会创建一个全局符号表，会话结束时则将其删除。不过，在特定条件下，宏执行期间会临时创建多个局部符号表，并在执行结束后将其删除。

3. R 中的惰性求值

惰性求值意味着只有在参数被使用时才会对其求值。在编程语言中实现惰性求值可以提高计算效率并降低内存消耗，尤其是在处理非常复杂的计算时，因为只有在需要时才会对参数求值。不过，有时这也会让程序员更难理解代码。

如 R 程序 1 所示，参数 y 依赖于 x，而参数 z 甚至可以由函数中稍后定义的变量（a 和 b）确定。惰性求值可以确保在定义 y 时不会使用 x = 5 对其求值，而是在局部环境中调用 y 时，使用 x = 2 才进行求值。

```
###R 中参数惰性求值的示例 惰性_求值<-函数
(x=5,y=x*10,z=a+b){   x=2a=3b=4c(x,y,z) } 惰性
[1] 2 20  7   求值() ###R 中的结果
```

R 程序 1 参数的惰性求值

3.1 Promise

R 中的惰性求值基于一种称为 Promise 的数据结构（Wickham，2019）。Promise 是一种特殊类型的对象，表示尚未求值的函数参数。它包含三个关键组成部分：

- 表达式：作为参数传入的实际 R 代码或表达式。
- 环境：表达式应在其中求值的上下文，其中包含求值所需的变量或函数。

- 值：初始时未设置；此部分用于存储表达式计算完成后的求值结果 d.

在 R 中调用函数时，每个参数都会被封装在一个 Promise 中，而不会立即求值。这样可以暂缓求值，直到函数体明确需要该表达式的值。一旦需要 Promise 的值，R 就会在其存储的环境中对表达式求值，并将结果缓存在 Promise 本身中。如果再次访问该 Promise，R 会使用缓存的结果，避免重复求值。Promise 数据结构的优点在于，它在创建时不占用任何内存空间；只有在求值后才会存储该值。这一特性可以提高计算效率并减少内存消耗，尤其是在处理非常复杂的计算时。

然而，无法使用 R 代码操作 Promise；它更像处于量子态的薛定谔的猫，任何试图用 R 代码检查它们的行为都会触发立即求值，使 Promise 消失（Wickham，2019）。要更好地理解 Promise 如何实现惰性求值，请参阅图2中的演示。

以 y 为例。调用函数 lazy_eval() 时，y 被赋予表达式 x*10，并初始化一个 Promise。该 Promise 同时封装了表达式 x*10 和函数执行环境。在 R 执行 c(x,y,z) 时，Promise y 会被求值以获取一个值；表达式 x*10 会在函数执行环境中求值，此时 x 已更新为 2，因此 y 的计算结果为 20。

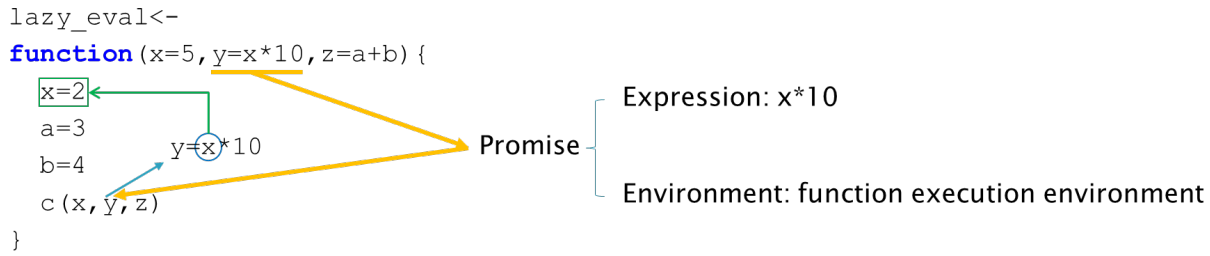

图2：Promise 数据结构如何实现 R 中的惰性求值

4. SAS 中的惰性求值

SAS 宏不会像 R 那样对默认参数实现惰性求值。不过，如果你在 SAS 宏中尝试类似的代码，就会得到与 SAS 程序 1 中相同的结果，这是为什么呢?

```
/*SAS 中参数“惰性求值”的示例*/
%宏惰性(x=5,y=&x*10,z=&a+&b);
%put _user_; /*这会在日志中显示符号表*/
%let x=2;
%let a=3;
%let b=4;
%put (&x %eval(&y) %eval(&z));
%mend;

%惰性求值()
(2 20 7)  /*SAS 日志中的结果*/
```

SAS 程序 1。SAS 参数的惰性求值

之所以能得到与 R 中惰性求值相同的结果，是因为 SAS 中存在局部符号表（Ling & Wang, 2025）。在 SAS 中，如图 3 所示，词法扫描器会读入 SAS 代码，对其进行标记化，然后传递给编译器。当词法扫描器在代码中检测到宏时，会自动触发宏处理器，后者会创建一个局部符号表，将参数作为宏变量存储起来（Lyons, 2004）。在我们的例子中，局部符号表会将 5 存储为 x 的值，将 ‘%eval(&x*10)’ 存储为 y 的值，并将 ‘%eval(&a+&b)’ 存储为 z 的值。

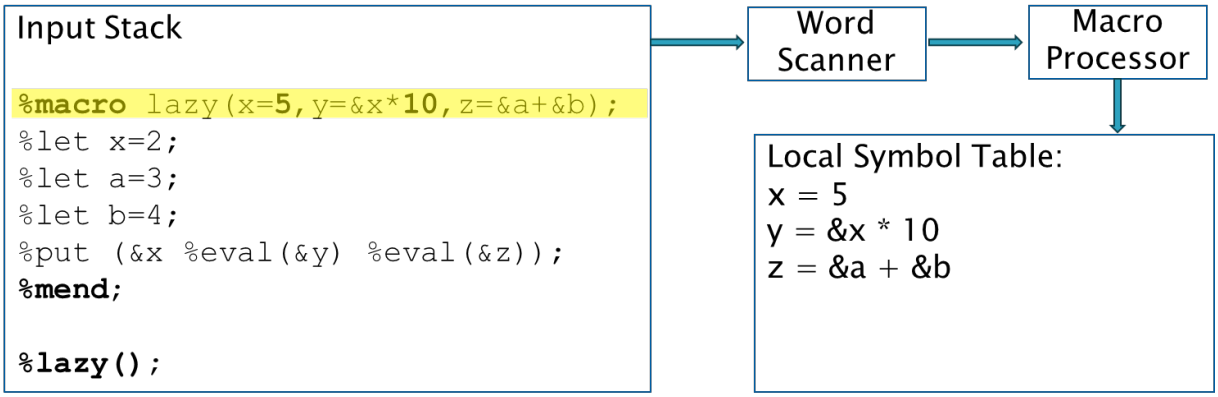

图 3. SAS 中惰性求值的演示 1

将宏语句中的参数存入局部符号表后，SAS 会继续读取宏内部的代码，将 x 的值更新为 2，并将 a、b 添加到局部符号表中。然后，在 %put (&x %eval(&y) %eval(&z))statement 中，SAS 会在局部符号表中查找并获取 x、y、z 的值，再将这些值写入日志。例如，首先会获取 y 的值，即 %eval(&x * 10)，然后也会获取并替换 x 的值，得到 %eval(2*10)。在接下来的执行过程中，日志中会写入 20，作为 y 的求值结果。

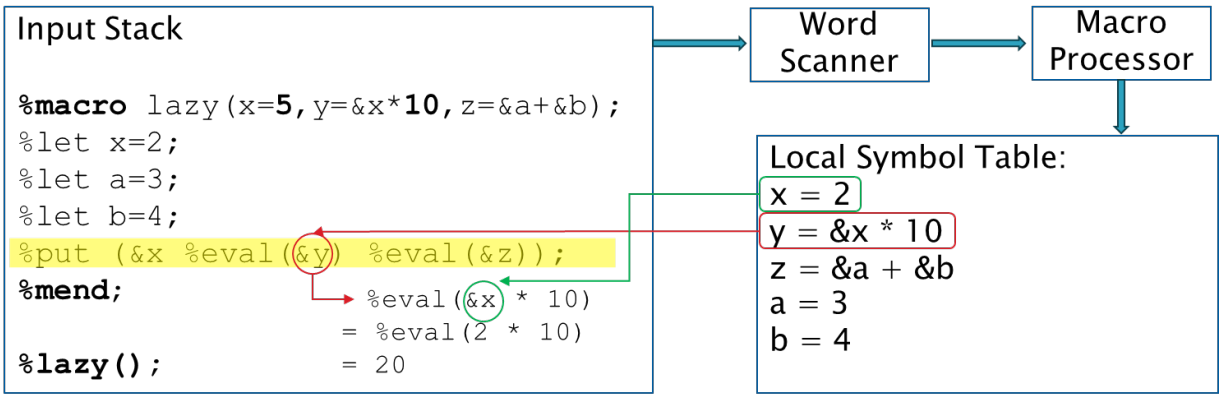

图 4. SAS 中惰性求值的演示 2

R 和 SAS 都会使用一个位置来存储默认参数/参数的表达式，但 R 中的 Promise 在被调用之前不会占用内存，而 SAS 会使用内存空间（局部符号表）来存储这些表达式。

5. 求值策略

从上述示例中我们了解到，SAS 和 R 都会对默认参数进行惰性求值，因此会得到相同的结果。然而，由于惰性求值机制不同，SAS 和 R 的求值策略并不相同，偶尔会导致不同的结果。

5.1 R 中的按需调用

R 采用由其 Promise 数据结构支持的按需调用策略。在这种范式中，表达式只有在明确需要其值时才会求值，因此只会求值一次，之后的使用则会读取缓存的结果。在 R 程序 2 中，y 的值被打印了两次，且两次的值相同。首次在 print() 函数中调用 y 时，Promise 会使用 x = 2 求值，返回 20 作为要打印的值。随后，这个 Promise 的值会存储为 y 的值。因此，之后即使环境中的 x 值发生变化，y 的值仍保持不变，因为 Promise 不会重新求值。

```
惰性1<-函数(x=5,y=x*10){
```

```
      x=2
print(y)
x=10
print(y) }
lazy1()
[1] 20
[1] 20
```

R 程序 2。R 中按需调用策略的示例

该策略通过消除冗余计算来提升资源优化效果，这对于数据密集型处理尤为有利。它还允许 R 函数有条件地执行代码，确保只进行必要的计算。不过，使用 Promise 时需要经过周密设计，以管理潜在的副作用和延迟的错误检测，因为表达式只有在被访问时才会求值。

5.2 SAS 中的按名调用

在 SAS 中，按名调用策略是其惰性求值方式的基础。采用按名调用时，表达式会延迟到执行流程中需要它们的时刻才求值。这意味着宏变量会以文本形式求值，并在运行时动态替换。在 SAS 程序 2 中，与 R 程序 2 类似，y 的值会被写入日志两次。第一次输出的 y 为 20，与 R 中的结果一致；第二次输出则变为 100。这是因为采用按名调用策略时，第一次求值的结果不会像 R 中那样 被缓存。当 SAS 随后执行 x = 10 时，局部符号表中的 x 值会更新为 10。因此，第二条 %put 语句会重新对 y 求值，得到使用 x = 10 时的 y = 100 。

```
%宏 lazy1(x=5,y=&x*10);
%let x=2;
%put %eval(&y);
%let x=10;
%put %eval(&y);

%mend;

%惰性求值
()
20
100
```

SAS 程序 2：SAS 中按名调用策略示例

尽管这种方法在操作变量名和构建灵活的代码结构方面提供了灵活性，但如果多次调用变量，可能会导致重复求值。因此，程序员可能需要仔细设计宏，以平衡灵活性和效率，并确保延迟求值不会带来不必要的开销。

6. 结论

本文比较了 SAS 和 R 的惰性求值机制，并着重指出了体现各自优势的关键差异。R 采用按需调用策略，由 Promise 数据结构提供支持，该结构会延迟计算，并且在求值之前不占用内存。这种方法显著提高了计算效率，减少了内存消耗，并简化了处理流程。相比之下，SAS 采用按名调用策略，利用符号表实现动态灵活的宏功能，非常适合复杂的数据操作任务。虽然这种方法具有适应性，但需要谨慎管理，以避免可能影响性能的重复求值。随着制药行业从 SAS 转向 R，了解这些策略有助于程序员在发挥 SAS 灵活性的同时利用 R 的效率。通过发挥两种语言各自的优势，开发人员可以改进编码实践，确保在各种分析任务中实现最佳性能和资源利用。

7. 参考文献

Ling, C. 和 Wang, Y 。（2025）。从动态到词法：SAS 与 R 中作用域规则的比较研究。SESUG

*2025*年会议论文集。 北卡罗来纳州卡里。

Ling, C.，& Wang, Y. (2025)。用于自动生成和验证表格、列表和图形的高兼容性 R Shiny 平台。*PharmaSUG 2025*年会议论文集。 加利福尼亚州圣迭戈。获取自：https://pharmasug.org/proceedings/2025/OS/PharmaSUG-2025-OS-076.pdf

Ling, C.、& Wang, Y. (2025)。用 R 编写 SAS 宏？R 函数可以帮上忙！*PharmaSUG 2025*年会议论文集。 加利福尼亚州圣迭戈。获取自：https://pharmasug.org/proceedings/2025/AP/PharmaSUG-2025-AP-078.pdf Lyons, L. (2004)。深入探究：宏处理器究竟如何运作？东北部 *SAS* 用户组。 马里兰州巴尔的摩。获取自 https://www.lexjansen.com/nesug/nesug04/pm/pm07.pdf Wickham, H. (2019)。高级 *R*（第 2 版）。获取自 https://adv-r.hadley.nz/index.html