arXiv:2601.17578v1 [cs.DC] 24 Jan 2026

使用 Futures 在 R 中实现并发、并行 Map-Reduce 的统一方法

作者：*Henrik Bengtsson*

摘要 R 生态系统为迭代计算提供了丰富多样的映射-归约应用程序编程接口（API），然而，要在这些不同的框架中实现代码并行化，必须学习多个通常彼此不兼容的并行 API。**futurize**软件包通过提供一个名为 futurize() 的单一函数来应对这一挑战；该函数可将顺序映射归约表达式转换为并行等价形式，而具体的繁重工作则由 future 生态系统完成。借助 R 原生管道运算符，用户只需进行极少的重构即可将现有代码并行化——通常只需在表达式后附加“|> futurize()”即可。该软件包支持来自基础 R、**purrr**、**crossmap**、**foreach**、**plyr**和 **BiocParallel** 的经典映射归约函数，例如 lapply(xs, fcn) |> futurize() 和 map(xs, fcn) |> futurize()；同时还支持不断增加的一组特定领域软件包，例如 **boot**、**caret**、**glmnet**、**lme4**、**mgcv** 和 **tm**。通过抽象底层并行机制，并统一 future 选项的处理方式，该软件包使开发者能够通过 futurize() 声明要并行化的内容，同时让最终用户通过 plan() 选择并行化方式。本文介绍了 **futurize** 的理念、设计和实现，展示了其在各种映射归约范式中的用法，并讨论了它在简化 R 中并行计算方面所发挥的作用。

1 引言

R 中一种常见的数据处理策略（R 核心团队，2025）是使用映射归约调用，即将一个函数应用于集合中的每个元素。这种映射归约范式也称为拆分-应用-合并策略（Wickham，2011），已在若干流行但各不相同的 API 中实现。基础 R 提供了 `lapply()` 及相关函数，**purrr** 软件包（Henry and Wickham，2025）提供了 `map()`等函数，而**foreach** 软件包（微软公司和 Weston，2022b）则引入了自己的 `foreach() %do% { }` 结构。它们各有优缺点，但最终都能达到相同的目标。正确使用时，它们会并发处理每次迭代[1]，且各次迭代彼此独立、不会产生副作用——这种特性称为“易于并行”，因此这类计算非常适合并发和并行执行。

并发处理对于组织计算密集型任务至关重要。随着数据集不断增大、分析日趋复杂，将工作分配到多个 CPU 核心或计算机上的能力也愈发重要。R 生态系统提供了丰富多样的并发与并行处理方法，其中许多方法问世至今已有十多年（Schmidberger 等人，2009；Eddelbuettel，2021年）。R 从 2.14.0 版（2011 年）起，通过 **并行** 软件包内置支持其中一些方法；此外，CRAN 和 Bioconductor 上也有社区贡献的并行映射归约 API。一些替代方案与原始 API 和行为非常相似，另一些则只有部分相似。例如，使用`parallel::mclapply()`时，与 `parallel::lapply()`不同，错误会在内部通过 `try()` 捕获，并以 `try-error`条件的形式返回，而不是发出错误信号；返回结果也不包含对原始错误对象的引用。同样，`parLapply()` 函数会捕获错误，且无法访问原始错误对象。另一方面，与 `mclapply()`相比，它确实会发出错误信号，并附带包含原始错误消息的消息字符串。二者都不会传递消息或警告，这些内容会完全丢失，标准输出（“stdout”）也是如此。从查看使用这些函数的 CRAN 和 Bioconductor 程序包源代码以及在线支持论坛可以看出，许多开发者和最终用户都认为这些函数的行为与 `lapply()` 相同，并不了解上述差异。据作者所知，future 生态系统（Bengtsson，2021年）是 R 中唯一一个开箱即能保留错误并传递输出和条件的并行化框架。

此外，许多并行映射-归约函数都有各自的子 API，用于微调并行化的执行方式，例如并行进程的数量及其类型。这种碎片化不仅增加了开发者不必要的认知负担，也带来了锁定问题。开发者通常必须花费大量时间评估各种替代方案，才能选择要支持的特定并行框架；而由于之后不愿再评估其他解决方案，锁定风险也随之增加。这种锁定会影响开发者今后的编码风格选择，也可能限制最终用户对代码执行方式和执行位置的选择。例如，使用 mclapply() 时，并行化仅限于 Unix 和 macOS，无法支持 Windows 用户。类似地，使用 parLapply() 配合 makeCluster()，或使用 foreach() %dopar% { }

[1]本文使用“并发”一词来描述计算的逻辑结构，并使用“并行”一词来描述其在多个资源上的物理执行。

并采用常见的 **doParallel** (微软公司和 Weston， 2022a) 设置时，并行化仅限于本地计算机。future 生态系统的核心设计理念是尽量减少开发者和最终用户受到的此类锁定，并使各种实现能够自动灵活适应未来的技术进步。本文介绍的 **futurize** 软件包（Bengtsson，2026b）正是这一核心设计策略的一部分。

用于并行求值的 future 概念于 20 世纪 70 年代末提出（Hibbard，1976 年；Hewitt 和 Baker，1977 年；Friedman 和 Wise，1978 年）。在 R 中，future 包（Bengtsson，2026a）基于这一概念，建立了一个用于并行和分布式处理的统一框架。在此基础上，出现了多个软件包，为常见的映射-归约 API 提供并行版本，例如：future.apply（Bengtsson，2025b）为基础 R 的 apply 函数提供支持，furrr（Vaughan 和 Dancho，2022）为 purrr 的替代函数提供支持，而 doFuture（Bengtsson，2025a）则为 foreach 的迭代结构提供支持。尽管这些基于 future 的扩展成功地为各自的 API 引入了并行化，但它们要求开发者学习并使用略有不同的函数名称和并行选项约定。例如，将 lapply() 并行化需要切换到 future.apply::future_lapply()，将 purrr::map() 并行化需要切换到 furrr::future_map()，而将 foreach::foreach() %do% { } 并行化则需要切换到 doFuture 的 foreach::foreach() %dofuture% { }。除了不同的函数名称外，每个软件包还拥有自己的一套选项规范，用于指定如何微调并行执行。尽管这些差异很小，但它们足以使不同软件包需要各自不同的文档，进而割裂开发者和用户群体，使他们无法从使用更加统一的 API 所带来的跨领域交流中受益。

该**futurize** 软件包通过提供一个统一函数`futurize()`来应对这些挑战，该函数可以并行化任何受支持的顺序映射归约表达式。开发者无需将函数调用替换为自定义的并行对应函数，而是可以保留熟悉的顺序代码，只需将其通过管道传递给`futurize()`即可；

```
library(futurize)
plan(multisession)

xs <- 1:100 ys <- lapply(xs, fcn) |>
futurize()
```

这种方法具有以下几项优势：

1. **最少的代码更改**：并行化只需在现有表达式中添加 `|> futurize()` 。2. **熟悉的代码**：原有的顺序逻辑仍然清晰可见且易于阅读，使注意力集中于映射归约逻辑，而不是并行化细节。3. **熟悉的行为**：标准输出和条件——包括消息、警告和错误——与原有顺序逻辑的行为相同。4. **统一接口**：同一个`futurize()` 函数可用于所有受支持的映射归约 API。5. **统一选项**：无论底层 API 如何，`futurize()` 函数都提供一种统一且一致的方式来指定并行选项。6. **后端独立性**：与 future 生态系统中的所有工具一样，并行后端的选择交由最终用户决定，例如`plan(multisession)`、`plan(future.mirai::mirai_multisession)`以及`plan(future.batchtools::batchtools_slurm)`。

本文介绍了**futurize**的理念、设计与实现。第 2 节讨论该软件包的理念和设计原则。第 3 节详细介绍其实现，包括转译机制和支持的 API。第 4 节展示结果，说明该软件包在各种映射-归约范式中的用法。第 5 节探讨该软件包在 future 生态系统中的作用及未来的发展方向。

2 futurize 的理念与设计

futurize **的设计**直接延伸了**future**框架的核心理念：清晰地区分开发者关注的问题（要并行化什么）与最终用户关注的问题（如何以及在哪里运行它）。虽然**future** 软件包提供了实现这一目标的底层 API（`future()`、`resolved()`和`value()`），但对于常见的映射-归约任务，直接使用它可能会显得冗长。成熟的高层 API（例如**future.apply** 和**furrr** ）提供了更便捷的函数，隐藏了部分并行化编排细节，但开发者仍需学习并使用不同于原有顺序版本的新函数名。

这个futurize软件包通过充当高层“转译器”来弥合这一差距。其核心理念是，开发者应当能够使用自己熟悉的顺序映射-归约函数编写代码，然后只需调用一个简单的函数，便可选择启用并行化。其目标是尽可能减轻将顺序代码转换为并行代码时所需的认知负担和思维上下文切换。

2.1 设计原则

futurize 的设计遵循若干核心原则，这些原则源自更广泛的 future 生态系统理念，同时也包括一些针对其作为转译层这一角色的额外原则。

关注点分离。 遵循 future 生态系统理念，futurize 严格区分并行化的内容（即并发执行什么）与并行化的方式。开发者和用户通过将表达式传入`futurize()`来指定要并行化的内容。并行后端（例如本地多 CPU 核心、集群、云端或高性能计算 (HPC) 调度器）的选择完全由最终用户通过 `plan()`控制。这种分离确保使用futurize编写的代码能够自动适用于任何当前或未来的并行后端 [2]。

最小化 API 接口面。 该软件包主要只公开一个函数 `futurize()`，负责处理所有转译工作。这种精简设计降低了学习门槛，也让软件包更容易采用。控制 future 求值的选项作为参数传递给 `futurize()`本身。

保留现有代码。 与那些要求重写顺序代码以使用并行函数的方法不同，futurize 允许原有的顺序逻辑保持不变。表达式 `lapply(xs,fcn) |> futurize()` 清楚地表明了正在执行的操作（`lapply()`），以及该操作可以并行化（`futurize()`）。这种透明性有助于代码审查和维护。

全局禁用/启用。 出于调试和性能分析的目的，可以通过调用 `futurize(FALSE)` 全局禁用 futurize。所有 `futurize()` 调用随后都会直接传递，就好像 `|> futurize()` 不存在一样。调用 `futurize(TRUE)` 即可重新启用。需要特别注意的是，只有最终用户可以切换此设置；程序包绝不能全局禁用或启用 futurization。

2.2 转译方法

“转译”一词描述的是将源代码从一种形式转换为另一种形式的过程，也称为源代码到源代码转换（Loveman，1977 年；Aho 等，2006 年）。在 futurize中，转译会将顺序映射-归约表达式转换为逻辑等价的表达式，借助 future 生态系统实现并发执行。当 `futurize()` 接收到诸如`lapply(xs, fcn)` 这样的表达式时，它会捕获未求值调用，并识别映射-归约调用及其主函数。根据这些信息，f uturize 会将该表达式改写为基于 future 的对应形式。例如，

- base::lapply() |> futurize() 转译为future.apply::future_lapply()
- `purrr::map() |> futurize()` 转译为`furrr::future_map()`，并且
- `foreach::foreach() %do% { } |> futurize()` 转译为`foreach::foreach() %dofuture%{ }`，其中`%dofuture%` 属于 doFuture。

这种方法比简单的函数替换更加灵活，因为转译能够处理不同 API 的细微差异，包括参数命名和默认值方面的差异。通过`futurize()` 指定的并行化选项（也称 future 选项）会恰当地映射到底层实现，使开发者无论使用哪种映射-归约 API，都能使用统一的接口。

重要的是，转译方法将面向用户的语义与实现细节解耦。此外，这种分离使底层实现能够透明地演进。例如，尽管目前 `lapply()` 的 futurize 依赖于 future.apply，但未来版本的futurize可能会直接转译为仅使用核心 future API 的调用，而不会影响现有用户代码，也不会改变 futurize 的使用方式。

通过结合源代码到源代码转换与领域特定语言（DSL）原则（Fowler，2010），futurize 以简洁而富有表现力的语法，为现有 R 代码引入并发，同时保留其可读性、可组合性和长期稳定性。

[2]为了确保使用 future 的代码能够与任何 future 后端配合使用，future 后端必须符合 Future API。合规性通过 future.tests 软件包（Bengtsson，2025f）进行验证。

2.3 与 R 管道运算符集成

futurize 中的一个关键设计决策，是将其与 R 的原生管道运算符 |>集成。该运算符在 R 4.1.0 中引入，使“计算此项，但以并发方式进行”这一意图能够自然地表达出来：

```
ys <- lapply(xs, fcn) |> futurize()
```

这种语法将并行化修饰符放在表达式末尾，使核心逻辑更加突出。根据原生管道运算符的定义，`expr |> futurize()` 与`futurize(expr)`完全相同，也就是说，上面的示例也可以写成 `ys <- futurize(lapply(xs,fcn))`，但出于可读性考虑，优先使用管道语法。

2.4 统一的 future 选项

futurize 的主要目标之一，是通过`futurize()` 函数提供一种统一的方式来指定并行选项。底层的并行程序包future.apply、furrr和doFuture在指定 future 选项（例如随机数生成和全局变量处理）以及控制负载均衡时，各自采用略有不同的约定。例如，fu-ture.apply使用`future.seed`、`future.scheduling`等参数；furrr则使用`.options` 搭配`furrr_options()`；而doFuture的`%dofuture%` 则使用 `foreach()` 参数 `.options.future`。这些差异会在切换 API 时造成障碍，也会让相关文档变得不必要地复杂，令用户难以理解它们之间的差异。`futurize()` 函数将这些差异抽象起来，提供一个统一且一致的接口：

```
ys <- lapply(xs, fcn) |> futurize(seed =TRUE, chunk_size = 2)
```

主要选项包括：

- 随机种子：控制并行随机数生成。设为`TRUE`时，将使用 L’Ecuyer-CMRG 流（L’Ecuyer, 1999；L’Ecuyer 等人，2017），确保并行工作进程生成的随机数可复现且具有良好的统计性质。对于某些函数，例如 `replicate()` 和`foreach::times()`，`futurize()` 的默认值为`seed = TRUE`。
- `globals`：控制全局变量的识别和导出。默认情况下，通过静态代码分析自动识别全局变量（Bengtsson，2025g；Tierney，2024年）。
- 程序包：指定要在工作进程上加载的程序包。默认情况下，会根据已识别的全局变量自动推断所需程序包。
- 标准输出和条件：控制对工作进程标准输出、消息和警告的捕获与转发。
- 调度和分块_大小：通过确定每个 future 处理的元素数量来控制负载均衡。

无论底层表达式使用的是`lapply()`、`purrr::map()`等，future 选项的作用都始终一致。这种统一显著简化了用户的思维模型，尤其是对于使用多个映射-归约 API 的用户而言。统一的设计也使得日后添加更多选项变得轻而易举。

3 实现

3.1 包依赖与要求

futurize软件包依赖 R 4.1.0 版本（2021 年 5 月）或更高版本，以支持原生管道运算符[3]。虽然严格来说并非必需，但出于实际使用目的，它依赖于 future，例如在执行 `library(futurize)` 后提供 `plan()`。所有其他包依赖都是可选的（“建议”依赖），也就是说，只有在需要对相关映射-归约调用进行 futurize 时才需要它们。例如，基础 R 映射-归约调用的 futurize 依赖于 future.apply，purrr 调用依赖于furrr，而foreach调用依赖于 doFuture。有关当前支持的函数及其包依赖，请参见表 1。这些“准入”包依赖会延迟加载；开发者和用户只需安装与其使用场景相关的软件包即可。如果使用这些包，软件包开发者需要导入它们——导入操作表明，使用这些包的软件包会通过`futurize()`将其作为自身的一部分进行安装。

[3]理论上，通过使用`futurize(expr)`，futurize 也能在低于 4.1.0 的旧版 R 中运行。不过，在 ord er 要在 CRAN 上发布futurize，并且 |> 在文档及其示例中使用它，需要依赖 R 4.1.0。

3.2 转译过程的实现

本节介绍 **futurize** 如何实现第 2.2 节概述的转译方法。该过程分为以下几个步骤：

1. **表达式捕获**：当一个表达式被传入`futurize()`时，该函数会使用`substitute()`捕获未求值调用。这是 R 中一种成熟的非标准求值（NSE）技术（Cham-bers，2008；Wickham，2019），可让函数将代码作为数据进行处理。2. **函数识别**：随后，它会检查该调用，以识别被调用的函数。它会确定函数名称及其所属的命名空间，这对于区分不同程序包中同名的函数至关重要。3. **转译器查找**：该软件包为已知函数维护着一个内部转译器注册表，并会查询该注册表，查找与已识别函数相匹配的转译器。4. **表达式重写**：找到正确的转译器后，它会重写原始表达式。转译器会将原始调用中的参数映射到目标函数中对应的参数，并传递通过`futurize()`指定的并行化选项。5. **求值**：最后，在原始环境（父框架）中对新构造的表达式求值。

要查看不同的调用如何被转译，请使用 `futurize(eval = FALSE)`。这会返回转译后的调用，而不对其求值。

3.3 表达式解包

出于实际考虑，转译机制包含用于“解包”由 `{}`、`()`、`local()`、`I()`、`identity()`、`suppressMessages()` 和 `suppressWarnings()` 等结构包裹的表达式的逻辑。转译器会逐层深入这些包裹结构，直到找到可转译的表达式，因此无需在此类结构中放置`futurize()`。这样便可以采用如下模式：

```
ys <- { lapply(xs, fcn) } |>
suppressMessages() |>futurize()
```

从而避免必须这样写：

```
ys <- { lapply(xs, fcn) |> futurize() }
 |> suppressMessages()
```

在讨论进度报告时，第 4.10 节给出了一个更复杂的展开示例。

3.4 支持的 API

该**futurize** 软件包支持对多个程序包中的函数进行转译。表 1 和表 2 分别汇总了受支持的映射-归约函数和领域专用函数。

表 1： 当前支持并行转译的映射-归约函数包括`futurize()`。

软件包	函数	依赖
base	lapply()、sapply()、tapply()、vapply()、mapply()、.mapply()、Map()，eapply()，apply()，by()，replicate()，Filter()	**future.apply**
stats	kernapply()	**future.apply**
purrr	`map()` 及其变体、`map2()` 及其变体、`pmap()` 及其变体，imap() 及其变体、modify()、modify_if()、modify_at()，map_if(), map_at(),invoke_map()	**furrr**
crossmap	xmap() 及其变体，xwalk()，map_vec()，map2_vec()，pmap_vec()，imap_vec()	（本身）
foreach	%do%，例如，foreach() %do% { }，times() %do% { }	**doFuture**

软件包	函数	依赖
plyr	aaply() 及其变体，ddply() 及其变体，llply() 及其变体，mlply() 及其变体	doFuture
BiocParallel	bplapply()，bpmapply()，bpvec()，bpiterate()，bpaggregate()	doFuture

表 2： 用于并行转译的futurize() 当前支持的领域专用函数。

软件包	函数	依赖
boot	boot()、censboot()、tsboot()	future
caret	bag()，gafs()，nearZeroVar()，rfe()，safs()，sbf()，train()	doFuture
glmnet	cv.glmnet()	doFuture
lme4	allFit()，bootMer()	future
mgcv	bam()，predict.bam()	future
tm	TermDocumentMatrix()，tm_index()，tm_map()	future

要查看目前支持哪些程序包，请使用：

```
futurize_支持的_软件包() #> [1]"base" "BiocParallel" "boot" "caret" "crossmap" #
> [6]"foreach" "glmnet" "lme4" "mgcv" "plyr" #> [11]"purrr" "stats" "tm"
```

若要查看特定程序包支持哪些函数，请使用：

```
futurize_支持_函数("caret")
#> [1] "bag" "gafs" "nearZeroVar" "rfe" "safs" #> [6] "sbf" "train"
```

4 结果

本节展示 **futurize** 在各种映射-归约和领域特定范式中的用法，着重说明其统一的并行化方法。映射-归约示例涵盖基础 R、**purrr**、**foreach**和**plyr** （Wickham，2023年）。领域特定示例涵盖以下程序包：**boot** （Canty 和 Ripley，2024年）、**lme4** （Bates 等，2024年）、**glmnet** （Friedman 等，2024年）和**caret** （Kuhn，2024年）。为便于阅读，大多数示例都经过了大幅简化，但仍展示了futurize()如何应用于不同类型的映射-归约调用。这些示例并非旨在展示现实世界中的用例。对于所有示例，我们假设用户已注册并行后端：

```
library(future)
plan(multisession)
```

此设置会利用 **parallel** 软件包的 PSOCK 并行化，在本地机器上使用并行工作进程。事实上，[4] 此处的示例可使用 **future** 框架支持的任何兼容后端，而无需更改代码，包括**future.callr** (Bengtsson，2025d；Csárdi 和 Chang，2024年）、**future.mirai**(Bengtsson，2025e；Gao，2025），以及通过**future.batchtools**(Bengtsson，2025c；Lang 等人，2017）使用高性能计算调度器。

4.1 使用基础 R 的基本用法

一个常见的用例是并行化 lapply() 调用。例如，考虑一个执行耗时计算的函数：

```
慢速_函数 <- function(x) {
Sys.sleep(1.0) # 模拟工作 x^2}
```

[4]参见https://www.futureverse.org/backends.html，了解兼容的 future 后端。

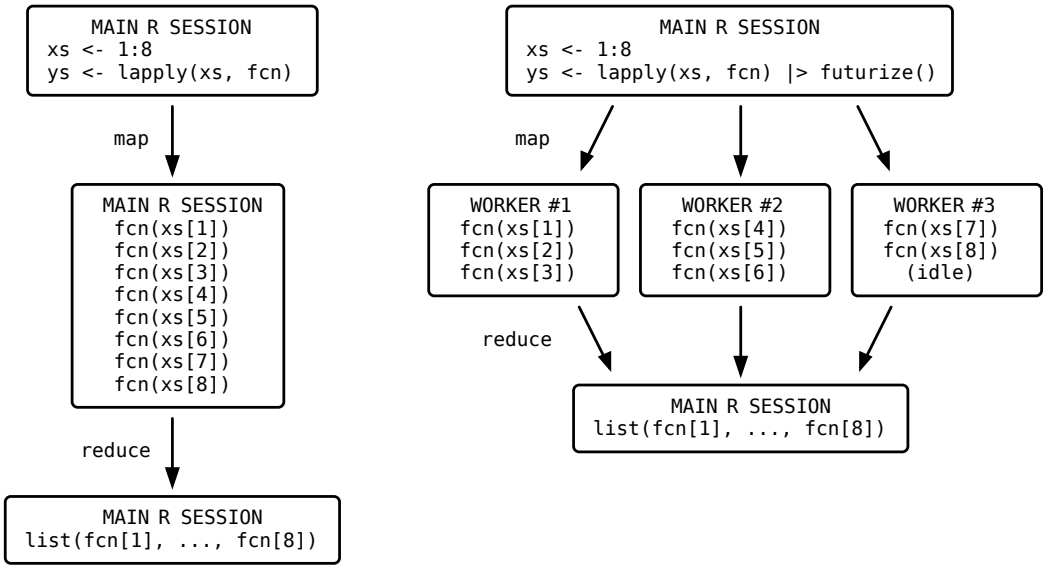


图 1： 示意图展示了映射-归约`lapply()` 调用在不使用`futurize()` （左）和使用`futurize()` （右）时的求值方式。总共有八次`fcn()` 调用——输入向量`xs`中的每个元素对应一次。默认情况下，这些函数调用（即“任务”）会依次求值。使用`futurize()`后，求值由 future 生态系统负责处理。例如，使用三个并行工作进程时，任务会分配给各个工作进程并发处理，之后收集结果并归约为最终形式。根据 future`plan()` 的设置，并行工作进程可以在当前机器本地运行，也可以在外部机器上远程运行。对`purrr::map()` 和`foreach::foreach()%do%{ }`进行 futurize 时，并行调度的工作方式相同。

当我们使用以下方式将此函数应用于 100 个元素时：

xs <- 1:100 ys <- lapply(xs,
slow_fcn)

需要 ~100 秒才能完成。在发现流水线中的这一瓶颈，并用尽提升 `slow_fcn()` 本身性能的办法后，自然会转而采用能够并发求值 `slow_fcn(xs[1])`、……、`slow_fcn(xs[100])` 的解决方案，以缩短总体周转时间（“墙钟时间”）。借助上述本地“多会话”工作进程，我们可以使用以下方式实现这一点：**futurize**：

xs <- 1:100 ys <- lapply(xs, slow_fcn) |>
futurize()

代码唯一的改动是在末尾添加 `|> futurize()`。计算逻辑保持不变。图 1 概述了在不使用和使用 `futurize()` 处理的情况下，映射-归约的求值方式。

除了`vapply()` 和`tapply()`等传统映射-归约函数之外，还有`replicate()`这样的函数，可用于多次求值同一个 R 表达式。它可以并行化如下：

样本 <- replicate(100, rnorm(10)) |> futurize()

由于`replicate()` 主要用于重抽样（例如置换检验），而这需要随机数生成器，`futurize()` 在检测到replicate()时，默认采用futurize(seed = TRUE)。这可确保无论使用哪种 future 后端（Bengtsson，2021年），生成的随机数在统计上均可靠、可复现且相互独立。

4.2 使用 ‘purrr’

purrr 软件包提供了一套广受欢迎的函数式编程工具包。其 `map()` 函数系列 s，它与基础 R 中的对应功能密切相关，并可与**futurize**无缝实现并行，例如：

```
library(purrr)

xs <- 1:100

# 顺序
ys <- map(xs, slow_fcn)

# 并行
ys <- map(xs, slow_fcn) |>futurize()
```

更复杂的管道也能自然地运行。例如，在以下代码中：

```
ys <- 1:100 |>map(rnorm, n = 10) |> futurize(seed =
TRUE) |>map_dbl(mean) |> futurize()
```

两个“map”调用都会并行化，且第一个调用会进行适当的随机数处理。

4.3 使用‘foreach’

foreach 软件包提供了另一种迭代范式。它因可选支持通过不同的“foreach适配器”并行处理迭代而广受欢迎。使用**futurize**，可以直接并行化`%do%` 运算符：

```
library(foreach)

xs <- 1:100

# 顺序
ys <- foreach(x = xs) %do% { slow_fcn(x) }

# 并行
ys <- foreach(x = xs) %do% { slow_fcn(x) } |> futurize()
```

尽管使用了 **foreach**，但无需将 `%do%` 改为 `%dopar%` 或 `%dofuture%`，也无需注册 foreach 适配器——这些都由`futurize()` 转译器抽象处理。

接下来，`times()` 函数与基础 R 中的 `replicate()` 类似，也可以正常使用，例如：

```
samples <- times(100) %do% rnorm(10) |> futurize()
```

出于与使用`replicate()` 时相同的原因，使用`times()` 时，`futurize()` 默认采用`futurize(seed = TRUE)`。

此外，**iterators** 软件包（Revolution Analytics 和 Weston，2022）中的迭代器与**foreach** 软件包紧密结合，也能按预期工作，例如：

```
library(iterators)

df <- data.frame(a = 1:4, b =letters[1:4])

ys <- foreach(d = df, i = icount()) %do%
{ list(value = d, index =i) } |> futurize()
```

4.4 使用 ‘plyr’

plyr 软件包提供了大量映射-归约函数。尽管该软件包已正式停止维护，但仍在维护中且广泛使用。它的 `llply()` 函数与`lapply()`和`purrr:map()`非常相似，可以并行化如下：

```
library(plyr)
```

```
xs <- 1:100

# 顺序 ys <- llply(xs, slow_
fcn)

# 并行 ys <- llply(xs, slow_fcn) |> f
uturize()
```

4.5 其他映射-归约软件包

The **crossmap** 软件包 (Hayes, 2025) 扩展了 **purrr** 函数集。例如，`xmap()` 可将函数应用于列表中元素的每种组合。借助 **futurize**，其函数可以并行执行。对于 Bioconductor 用户，**futurize** 可与 **BiocParallel** (Wang 等人, 2025) 无缝集成。BiocParallel 是 Bioconductor 的核心软件包之一。这使 Bioconductor 工作流能够利用 future 生态系统中的所有并行后端，包括 **future.callr**、**future.mirai**，以及通过 **future.batchtools** 使用的 HPC 调度器。有关这些软件包的更多详情和示例，请参阅 **futurize** 的软件包指南。

4.6 特定领域示例

使用 'boot' 软件包进行自助法

boot 软件包是"推荐"R 包之一，这意味着它得到 R 核心团队的官方认可、维护良好、稳定性高，并且默认随 R 一同安装。该软件包可生成自助法样本，并提供相关统计方法。其核心函数 `boot()` 可针对数据计算某个统计量的自助法样本。鉴于其重抽样特性，该算法非常适合并行化。`boot()` 函数内置了并行化机制，可通过不同组合使用参数 `parallel`、`ncpus` 和 `cl`。例如，要使用四个工作进程进行并行计算，可以这样做：

```
library(boot)
集群 <- 并行::makeCluster(4)

# 对 1920 年（'x'）和 1930 年（'u'）美国 49 个大城市的总体抽取 999 个
样本。对于每个样本，计算 # 1930 年平均人口与 1920 年平均人口之比 <-
function(pop, w) sum(w * pop$x) / sum(w * pop$u) b <-
boot(bigcity, statistic = ratio, R = 999, stype ="w", parallel = "
snow", ncpus = length(cl),cl = cl)
```

只有检查`boot()`的源代码后，我们才会发现，参数`ncpus` 必须严格大于 1，`parallel = "snow"` 和`cl = cl` 才会生效，这意味着默认值`ncpus = getOption("boot.ncpus", 1L)` 可能并不足够[5]。对于希望实现代码并行化的开发者而言，这又增加了一道额外门槛。**futurize**软件包通过隐藏以下细节来简化这一过程：

```
b <- boot(bigcity, statistic = ratio, R = 999, stype = "w") |> futurize()
```

使用 'lme4' 软件包的混合效应模型

lme4 软件包用于拟合混合效应模型。与 `boot::boot()` 类似，其`allFit()` 函数内置了通过不同组合的参数 `parallel`、`ncpus` 和 `cl` 实现并行化的机制。与 `boot::boot()` 类似，必须正确设置这三个参数，`allFit()` 才能通过 PSOCK**parallel** 集群实现并行化。**futurize** 软件包隐藏了这些细节，使我们能够通过管道传递给`futurize()`来实现并行化，例如：

```
library(lme4)

# 使用以下数据拟合广义线性混合效应模型（广义线性混合模型）：
```

[5]通过检查 `boot()` 的代码，我们发现任何大于 1 的 处理器数量 值都足够，并且给出相同的 e
结果，也就是说，我们也可以使用一个常数值，例如，处理器数量 = 2L。

```
# 牛传染性胸膜肺炎（CBPP）的数据，包含 56 个观测值和变量“发病数”“规模”
和“时期” gm <- glmer(cbind(incidence, size - incidence) ~period + (
1 | herd), data = cbpp, family = binomial) gm_all <- allFit(gm) |>
futurize()
```

使用“glmnet”软件包进行交叉验证

该**glmnet** 软件包采用高度优化的逐路径坐标下降算法，高效计算惩罚广义线性模型（套索回归、岭回归、弹性网络）的完整正则化路径。其`cv.glmnet()` 函数用于执行交叉验证。通过设置参数`parallel = TRUE`、启动并行工作进程并注册相应的**foreach**适配器，即可并行执行。使用**futurize**软件包可以简化这一过程：

```
library(glmnet)

# 模拟 1,000 个观测值和 100 个连续变量 n <- 1000 p <- 100 x <-
matrix(rnorm(n * p), nrow = n, ncol = p)


# 模拟 1,000 个连续响应变量 y <- rnorm(n)


# 通过交叉验证拟合广义线性模型 cv <-
cv.glmnet(x,y) |> futurize()
```

使用“caret”软件包进行分类与回归训练

该**caret** 软件包提供了丰富的机器学习工具，并配备统一的 API（Kuhn，2008）。`train()` 函数用于拟合模型，并可通过将参数`parallel= TRUE`、设置并行工作进程以及注册相应的**foreach** 适配器来支持并行处理。 **futurize**软件包将这一过程简化为：

```
library(caret)

# 使用具有 150 个观测值和五个变量的“iris”数据，通过 100 折交叉验证拟合随机森林模型 ctrl <-
trainControl(method = "cv", number = 100) model <- train(Species ~ ., data = iris,
model ="rf", trControl = ctrl) |> futurize()
```

4.7 其他领域专用软件包

mgcv软件包（Wood，2011）是 R 中其他“推荐”软件包之一。它提供了用于拟合广义加性模型（GAM）的方法，例如，`bam()` 函数可用于为包含数千个观测值的大型数据集（“大型加性模型”）拟合 GAM。它通过设置**并行**集群并将其作为参数`cluster`传入来支持并行处理，而这些细节由**futurize**抽象处理。 **tm** 软件包（Feinerer and Hornik，2024年）提供了各种文本挖掘方法。例如，映射归约函数`tm_map()` 可用于转换文本语料库，而**futurize** 可用于将其并行化，并隐藏诸如`tm_parLapply_engine()`等软件包特定的细节。有关这些软件包的更多详细信息和示例，请参阅**futurize** 的软件包指南。

4.8 后端灵活性

future 生态系统方法的一项关键优势是后端独立性。相同的 **futurize** 代码可在所有兼容的 future 后端中运行，因此能够实现并发执行；具体是并行、远程还是顺序执行，取决于后端。这意味着，使用 **futurize** 的代码可以自动利用未来推出的任何新 future 后端。以下是一些常用 future 后端的示例：

```
# 顺序处理（默认）plan(sequential)


# 本地基于 PSOCK 的并行处理，工作进程数量与当前 R 会话可用的
CPU 核心数量相同 plan(multisession)


# 本地基于 PSOCK 的并行处理（四个工作进程）
plan(multisession, workers = 4)

# 本地基于 fork 的并行处理（Unix/macOS）
plan(multicore, workers = 4)

# 本地基于 callr 的并行处理
plan(future.callr::callr, workers = 4)

# 本地基于 mirai 的并行处理 plan(future.mirai::mirai_
multisession, workers = 4)

# 本地计算机上的临时集群 plan(cluster, workers =
c("n1", "n1","n2"))

# 本地和远程计算机上的临时集群 plan(cluster, workers = c("n1", "n2",
"server.example.org"))

# 通过 batchtools 使用 Slurm 集群
plan(future.batchtools::batchtools_slurm)
```

未指定并行工作进程的数量时，默认使用系统允许的数量[6]。

4.9 熟悉的标准输出和条件处理行为

在运行 R 代码时，能够查看和捕获 `cat()`、`message()` 等函数的输出，并处理条件，是大多数用户习以为常的事情，但很少有并行框架支持这些功能，即使支持也不完整。future 生态系统的核心设计之一，是将在并行工作进程上生成的输出原样传回父 R 会话。这包括 `cat()`、`print()` 和 `str()` 等函数产生的标准输出，以及通过 R 的条件框架由 `message()`、`warning()` 和 `stop()` 等函数产生的输出。使用 `futurize()` 时，这种行为会自动延续，例如：

```
library(purrr) library(futurize)
plan(future.mirai::mirai_多会话)


ys <- 1:4 |> map_dbl(\(x) {
message("x = ", x) sqrt(x) })
 |> futurize() #>x = 1#>x
 = 2#>x = 3#>x = 4




ys #> [1] 1.000000 1.414214 1.732051 2.000000
```

输出和条件会原样传递，也就是说，我们可以像顺序运行代码时那样捕获并处理它们。例如，我们可以像这样抑制上面的消息：

[6]请参阅 `help("availableCores", package = "parallelly")` (Bengtsson, 2026c)，了解 future 生态系统遵循哪些系统设置，例如 Linux 控制组（“CGroups”）设置（Menage 等人，2008）、HPC 调度器分配以及各种环境变量。这可以降低 R 代码过度使用 CPU 资源的风险，这在多用户和多服务系统上至关重要。

```
ys <- 1:4 |> map_dbl(\(x) { message("x
 = ", x) sqrt(x) }) |>
suppressMessages() |> futurize()
```

```
ys #> [1] 1.000000 1.414214 1.732051 2.000000
```

4.10 进度报告

使用**progressr**软件包（Bengtsson，2025h），可以近乎实时地报告并行计算的进度：

```
library(progressr)
handlers(global = TRUE)
```

```
xs <- 1:100 ys <- local({ p <-
progressor(along = xs)
lapply(xs, \(x) { p() slow_fc
n(x) }) }) |> futurize()
```

进度更新会以近乎实时的方式从工作进程传回主进程，并在此转发，以便为长时间运行的计算提供反馈。这之所以可行，是因为进度会作为 R 条件发出，而**future**软件包和大多数 future 后端都会即时转发这些条件。

还要注意`futurize()`如何展开表达式（第 3.3 节）：它会逐层深入两个被包装的表达式，先是`local()`，然后是`{ }`，以识别需要 futurize 的映射-归约`lapply()`调用。

5 讨论

5.1 在 future 生态系统中的作用

The **futurize** 软件包在 future 生态系统中占据独特的位置。虽然**future.apply**、**furrr**和**doFuture** 等软件包为特定 API 提供完整的并行实现，**futurize**则提供统一的入口，内部将任务委托给这些软件包。这种分层架构带来了多项优势：

1. **单一入口**：开发者和用户只需学习`futurize()` 函数，就能并行化任何受支持的 API。2. **统一选项**：`futurize()` 函数提供了一种统一的方式来指定并行选项，同时屏蔽底层软件包各自采用的不同约定。3. **共享实现**：底层软件包继续处理各自 API 的具体细节，从而避免代码重复。

引入`futurize()`后，从用户的角度来看，**future.apply**、**furrr**和**doFuture** 等软件包都成为实现细节。用户无需加载这些软件包，甚至不必了解它们，只需确保已安装即可。**futurize** 软件包负责无缝转换，让用户唯一需要做的决定就是选择 future 后端`plan()`。

5.2 给软件包开发者的建议

考虑**futurize** 的软件包开发者应当：

1. **并行化试金石测试**。确保要并行化的映射-归约调用中没有副作用或元素间依赖。一个简单的测试方法是反转处理顺序，例如，`rev(lapply(rev(xs), fcn))` 应当与`lapply(xs, fcn)`得到相同的结果，唯一例外是随机数的生成顺序会有所不同。

2. **使用多进程后端进行测试**。 验证代码在使用相互独立的并行工作进程时能否正确运行，例如：`plan(multisession)`。 3. **正确处理随机数**。 当计算涉及随机数生成时，请使用 `seed = TRUE`。future 生态系统会检测是否在未声明 `seed =TRUE`的情况下使用了随机数生成器，并发出警告——务必处理这些警告。4. **让用户控制后端**。 不要在程序包中设置 `plan()`；应由最终用户自行设置。如果你这样做，[7]，就会把用户锁定在特定的 future 后端，使他们无法利用其他后端。[8]

5.3 未来工作

2015年 **future** 软件包发布后，社区开始关注“futurize”功能。早期开发主要致力于确定一种“语法糖”语法，并探索使用自定义中缀运算符（例如 `%futurize%`）或与 **magrittr** 管道（`%>%`）集成的方案，因为当时尚未引入 R 原生管道（`|>`）。另一个设计难题是 futurize 转译器应采用集中式还是分散式托管。分散式模式更可取，因为它能尽量减轻维护负担并最大限度地提高灵活性，但事实证明，集中管理是最好的起步方式。不过，计划开发一个标准化 API，使第三方软件包能够实现自己的转译器；

通用 futurize 支持。 为程序包实现自己的 futurize 转译器制定标准，例如提供一个内部的、非导出函数 `futurize_transpiler()`，供 `futurize()`查找。这样，开发者便可在自己的程序包中添加`futurize()` 支持，而无需请求更新 **futurize** 软件包。在此功能可用之前，作者很乐意将这些转译器添加到**futurize** 软件包中。

未来开发的其他计划方向包括：

验证与故障排查。 提供一些机制，用于测试映射归约调用在并行化与未并行化、按顺序与随机顺序迭代的情况下，是否会产生相同的结果。

可观测性与性能分析。 提供一些机制，用于分析使用和不使用`futurize()`的映射归约调用，以及使用不同 future 后端时的性能，帮助开发者更好地了解并行化究竟能带来多少收益。

结构化并发。 在 **futurize**的语境下，结构化并发（Sustrik，2016年；Chen 和 You，2023年）是一个相对现代的术语，指将并发任务的生命周期限制在映射-归约结构内。future 生态系统已经支持错误和中断的传播，以及 future 的取消。在大多数方面，发生错误或中断时，同级 future 已会被取消。不过，这一点尚未得到全面保证；例如，在设置 future 的过程中发生用户中断时，可能无法始终取消已经启动的 future。

简化进度报告。 为**progressr** 添加一个转译器`progressify()`，以便在映射归约调用中注入进度报告代码。例如，第 4.10 节中的冗长代码示例将变为`ys <- lapply(xs, slow_fcn) |> progressify() |> futurize()`。

资源规范 API。 用于指定映射-归约调用所需资源的标准化方法，例如每次迭代和函数调用所需的内存、文件和挂载点访问权限，以及最低软件包版本。这样，`futurize()` 和 future 生态系统就可以据此判断哪些并行工作进程能够承担该任务（如果有的话）。如果可用内存有限，资源规范也可用于控制并行任务的数量。

6 总结

futurize软件包是对 **future** 生态系统一项筹划已久的增强功能，它为在 R 中并行化顺序映射-归约操作提供了统一且适合管道的接口。通过将表达式转译为等效的 future 生态系统表达式，它让开发者能够保留熟悉的顺序代码，同时实现并行执行。该软件包支持范围广泛的 API，涵盖基础 R、**purrr**、**crossmap**、**foreach**、**plyr**、**BiocParallel**以及特定领域的软件包。它继承了核心 future 生态系统的所有优势，包括原样传递标准输出、条件和错误，自动识别全局变量，以及支持并行随机数生成的方法。

[7]如果确实必须在函数内部设置 future 后端，请务必临时设置，例如 `with(plan(multisession), local = TRUE)`。这样可以避免覆盖用户可能已设置并在其他地方使用的 future 后端。[8]扫描 CRAN 后，我们发现有几个软件包会在函数内部设置 `plan(multisession)`。这很不理想，因为这些函数无法利用现代并行化后端，例如 `plan(future.mirai::mirai_multisession)`，也无法利用未来出现的 future 后端。

futurize 的另一项关键优势是通过 `futurize()`提供统一的选项接口，抽象掉底层并行程序包各自采用的不同选项约定。这意味着无论使用哪种 MapReduce API，开发者和用户都能使用一套统一且一致的选项。这也使得从`lapply()` 迁移到`purrr::map()`或反向迁移都很容易。

秉持 future 生态系统的理念，**futurize** 严格区分并行化的内容（由开发者通过 `futurize()` 控制）和并行化的方式（由终端用户通过 `plan()` 控制）。这种分离确保使用 **futurize** 编写的代码可在任何并行后端上运行，从本地多 CPU 核心处理到 HPC 集群和云计算平台皆可。简单的模式 表达式 `|> futurize()` 提供了一条直观的并行化路径，无需修改代码，即可从单个笔记本扩展到超级计算机。

7 致谢

我感谢 R 社区，以及十多年来从一开始就为 future 生态系统做出贡献、提供反馈、报告错误和提出功能需求的所有人。future 框架的开发受益于 R Consortium 基础设施指导委员会（ISC）和陈-扎克伯格倡议（CZI）基础开源软件（EOSS）计划提供的资助机制。

参考文献

A. Aho、M. Lam、R. Sethi 和 J. Ullman。《编译原理、技术与工具》。Addison-Wesley，2006 年。doi：10.5555/1177220。D. Bates、M. Maechler、B. Bolker 和 S. Walker。lme4：使用“Eigen”和 S4 的线性混合效应模型，2024 年。URL https://CRAN.R-project.org/package=lme4。R 包版本 1.1-38。H. Bengtsson。使用 future 构建 R 中并行与分布式处理的统一框架。《R Journal》，13(2)：208–227，2021年。doi：10.32614/RJ-2021-048。H. Bengtsson。doFuture：使用“future”包的 Future API 的通用 Foreach 并行适配器，2025a。URL https://CRAN.R-project.org/package=doFuture。R 包版本 1.1.3。H. Bengtsson。future.apply：使用 future 并行应用于元素的函数，2025b。URL https: //CRAN.R-project.org/package=future.apply。R 包版本 1.20.1。H. Bengtsson。future.batchtools：使用“batchtools”进行并行与分布式处理的 Future API，2025c。URL https://CRAN.R-project.org/package=future.batchtools。R 包版本 0.21.0。H. Bengtsson。future.callr：使用“callr”进行并行处理的 Future API，2025d。URL https://CRAN.R- project.org/package=future.callr。R 包版本 0.10.2。H. Bengtsson。future.mirai：使用“mirai”进行并行处理的 Future API，2025e。URL https://CRAN.R-project.org/package=future.mirai。R 包版本 0.10.1。H. Bengtsson。future.tests：Future API 后端测试套件，2025f。URL https://CRAN.R-project. org/package=future.tests。R 包版本 0.9.0。H. Bengtsson。globals：识别 R 表达式中的全局对象，2025g。URL https://CRAN.R-project.org/package=globals。R 包版本 0.18.0。H. Bengtsson。progressr：用于进度更新的包容性统一 API，2025h。URL https://CRAN.R- project.org/package=progressr。R 包版本 0.18.0。H. Bengtsson。future：人人可用的 R 统一并行与分布式处理，2026a。URL https: //CRAN.R-project.org/package=future。R 包版本 1.69.0。H. Bengtsson。futurize：通过一个神奇函数并行化常用函数，2026b。URL https: //CRAN.R-project.org/package=futurize。R 包版本 0.1.0。H. Bengtsson。parallelly：增强“parallel”包，2026c。URL https://CRAN.R-project.org/package=parallelly。R 包版本 1.46.1。A. Canty 和 B. Ripley。boot：自助法函数（最初由 Angelo Canty 为 S 编写），2024 年。URL https://CRAN.R-project.org/package=boot。R 包版本 1.3-32。

J. M. Chambers。《使用 R 进行数据分析编程的软件》。施普林格，纽约；伦敦，2008 年。doi：10.1007/978-0-387-75936-4。Y.-A. Chen 和 Y.-P. You。结构化并发：综述。载于第 51 届国际并行处理会议研讨会论文集，ICPP '22 研讨会，纽约州纽约市，2023年。美国计算机协会。doi：10.1145/3547276.3548519。G. Csárdi 和 W. Chang。callr：在 R 中调用 R，2024 年。URL https://CRAN.R-project.org/package=callr。R 包版本 3.7.6。D. Eddelbuettel。使用 R 进行并行计算：简要综述。WIREs 计算统计学，13(2)：e1515，2021年。doi：10.1002/wics.1515。I. Feinerer 和 K. Hornik。tm：文本挖掘包，2024 年。URL https://CRAN.R-project.org/package=tm。R 包版本 0.7-7。M. Fowler。《领域特定语言》。Addison-Wesley，2010 年 10 月。ISBN 978-0131392809。D. P. Friedman 和 D. S. Wise。并行处理中应用式编程的若干方面。IEEE 计算机汇刊，C-27(4)：289–296，1978 年 4 月。doi：10.1109/tc.1978.1675100。J. Friedman、T. Hastie、R. Tibshirani、B. Narasimhan、K. Tay、N. Simon 和 J. Yang。glmnet：套索回归与弹性网络正则化广义线性模型，2024 年。URL https://CRAN.R-project.org/ package=glmnet。R 包版本 4.1-10。C. Gao。mirai：R 的极简异步求值框架，2025年。URL https://CRAN.R-project.org/package=mirai。R 包版本 2.5.3。A. R. Hayes。crossmap：将函数应用于列表元素的所有组合，2025年。URL https://CRAN.R- project.org/package=crossmap。R 包版本 0.4.3。L. Henry 和 H. Wickham。purrr：函数式编程工具，2025年。URL https://CRAN.R-project. org/package=purrr。R 包版本 1.2.0。C. Hewitt 和 H. G. Baker。通信并行进程定律。载于 IFIP 大会，页码 987–992，1977 年。URL https://dblp.uni-trier.de/db/conf/ifip/ifip1977.html#HewittB77。P. Hibbard。并行处理设施。载于 S. A. Schuman 编，《算法语言的新方向》。IRIA，1976 年。M. Kuhn。使用 caret 包在 R 中构建预测模型。《统计软件杂志》，28(5)：1–26，2008 年。doi：10.18637/jss.v028.i05。M. Kuhn。caret：分类与回归训练，2024 年。URL https://CRAN.R-project.org/ package=caret。R 包版本 7.0-1。M. Lang、B. Bischl 和 D. Surmann。batchtools：用于在批处理系统上工作的 R 工具。《开放源代码软件杂志》，2(10)：135，2017 年 2 月。doi：10.21105/joss.00135。P. L'Ecuyer。组合多重递归随机数生成器的优良参数与实现。运筹学，47(1)：159–164，1999 年。doi：10.1287/opre.47.1.159。D. B. Loveman。通过源到源转换改进程序。ACM 学报，24(1)：121–145，1977 年 1 月。ISSN 0004-5411。doi：10.1145/321992.322000。P. L'Ecuyer、D. Munger、B. Oreshkin 和 R. Simard。并行计算机的随机数：要求与方法，重点讨论 GPU。数学与计算机模拟，135：3–17，2017 年。ISSN 0378-4754。doi：10.1016/j.matcom.2016.05.005。专题：第九届 IMACS 蒙特卡罗方法研讨会。P. Menage、P. Jackson 和 C. Lameter。控制组，2008 年。URL https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt。Linux 内核文档。微软公司和 S. Weston。doParallel：适用于"parallel"包的 Foreach 并行适配器，2022a。URL https://CRAN.R-project.org/package=doParallel。R 包版本 1.0.17。微软公司和 S. Weston。foreach：提供 Foreach 循环结构，2022b。URL https: //CRAN.R-project.org/package=foreach。R 包版本 1.5.2。R 核心团队。R：统计计算语言与环境。R 统计计算基金会，奥地利维也纳，2025年。URL https://www.R-project.org/。

Revolution Analytics 和 S. Weston。*iterators*：提供迭代器构造，2022年。网址https://CRAN。 R-project.org/package=iterators。R 软件包版本 1.0.14。

M. Schmidberger、M. Morgan、D.Eddelbuettel、H.Yu、L.Tierney 和 U.Mansmann。使用 R 进行并行计算的最新进展。《统计软件杂志》，31(1):1–27，2009年。doi：10.18637/jss.v031.i01。

M. Sustrik。结构化并发，2016年。网址 https://www.250bpm.com/p/structured-concurrency。博客文章和设计说明。

L. Tierney。*codetools*：*R* 代码分析工具，2024年。网址 https://CRAN.R-project.org/package=codetools。R 软件包版本 0.2-20。

D. Vaughan 和 M. Dancho。*furrr*：使用 *future* 并行应用映射函数，2022。网址 https://CRAN.R-project.org/package=furrr。R 软件包版本 0.3.1。

J. Wang、M. Morgan、V. Obenchain、M. Lang、R. Thompson 和 N. Turaga。《*BiocParallel*：用于并行求值的 *Bioconductor* 工具》，2025年。网址：https://bioconductor.org/packages/BiocParallel。R 包版本 1.44.0。

H. Wickham。用于数据分析的拆分-应用-合并策略。《统计软件杂志》，40(1)：1–29，2011年。doi：10.18637/jss.v040.i01。

H. Wickham。《*Advanced R*，第二版》。《R 系列》。Chapman & Hall/CRC，CRC Press，2019年。doi: 10.1201/9781351201315。

H. Wickham。《*plyr*：用于数据拆分、应用与合并的工具》，2023年。网址：https://CRAN.R-project.org/package=plyr。R 包版本 1.8.9。

S. N. Wood。半参数广义线性模型的快速稳定限制最大似然和边际似然估计。《皇家统计学会杂志（*B*辑）》，73(1):3–36，2011。doi: 10.1111/j.1467-9868.2010.00749.x。

Henrik Bengtsson 加利福尼亚大学旧金山分校 流行病学与生物统计学系 旧金山，加利福尼亚州 *94143*，美国
ORCiD：
*0000-0002-7579-5165*henrik.bengtsson@gmail.com