arXiv:2303.02205v2 [cs.MS] 1 May 2024

Python 与 C++的奇妙世界

ManasviGoyal[1],**Ianna Osborne**[2],**Jim Pivarski**[2] 德里理
工大学，德里，印度[2] 普林斯顿大学，美国新泽西州普林斯顿 08544

电子邮件：mg.manasvi@gmail.com

摘要。将 Python 与 C++ 绑定，充分利用两种语言各自的优势，显然有诸多益处。这对于高能物理及其他科学界尤为重要，因为这些领域已大量投入 C++ 框架，并正迅速将数据分析迁移到 Python。作为 Scikit-HEP 的 Python 库，Awkward Array 第 2 版引入了一组仅头文件的 C++ 库，这些库不依赖任何应用程序二进制接口。用户可以在编译时直接包含这些库，而不必链接到特定平台的库。这项新进展使 Awkward Arrays 更容易、更便捷地集成到其他项目中，因为其实现很容易与 Awkward Array 代码库的其余部分分离。代码十分精简；它不包含在 Python 中使用 Awkward Arrays 所需的全部代码，也不引用 Python 或 pybind11。C++ 用户可以用它创建数组，然后将其复制到 Python，而无需使用任何专用数据类型——只需原始缓冲区、字符串和整数。这段 C++ 代码也简化了在 ROOT 中进行即时编译（JIT）的流程。这种实现方法解决了一些缺陷，例如原生依赖项可能带来挑战的项目打包问题。本文展示了如何采用仅头文件方法集成 C++ 与 Python。我们还介绍了新型 LayoutBuilder 和 GrowableBuffer 的实现。此外，本文还探讨了如何将 C++ 数据封装为 Awkward Arrays，以及如何在不复制数据的情况下将 Awkward Arrays 暴露给 C++ 。

1. 引言

Awkward Array [1] 是高能物理（HEP）社区在 Python 中进行物理分析的重要工具。它是 Scikit-HEP [2] 生态系统的一部分。嵌套可变长度列表（“参差不齐”或“锯齿状”数组）、包含不同类型字段的记录、缺失数据以及其他异构数据（联合／变体类型），都可以使用一组原语，并通过类似 NumPy 的 [3] Python [4]表达式进行定义。在 Awkward 数组中，面向用户的单个 ak.Array 由一棵小型树组成，每个节点都附有大型连续数据缓冲区 [5], ，如图 1 所示。编译操作在这些数据缓冲区上执行，而不是在它们所表示的对象上执行。

在这项工作中，我们介绍了用于在 C++中创建 Awkward Arrays 的新工具。此前，主代码库使用 C++ [6] 编写，其设想是让下游代码链接到 libawkward.so，但这条路径充满了隐藏问题 [7]。采用一个小型的仅头文件库，仅为下游代码填充数组缓冲区，再由下游代码使用 ctypes 将数据从 C++ 传递到 Python，这种方法更具前景。

2. Python-C++集成

如今，越来越多的前端用户使用 Python [8], ，但大规模处理仍需要 C++ [7] 的高性能。这就是我们将 Python 和 C++ 结合起来的原因，以便发挥

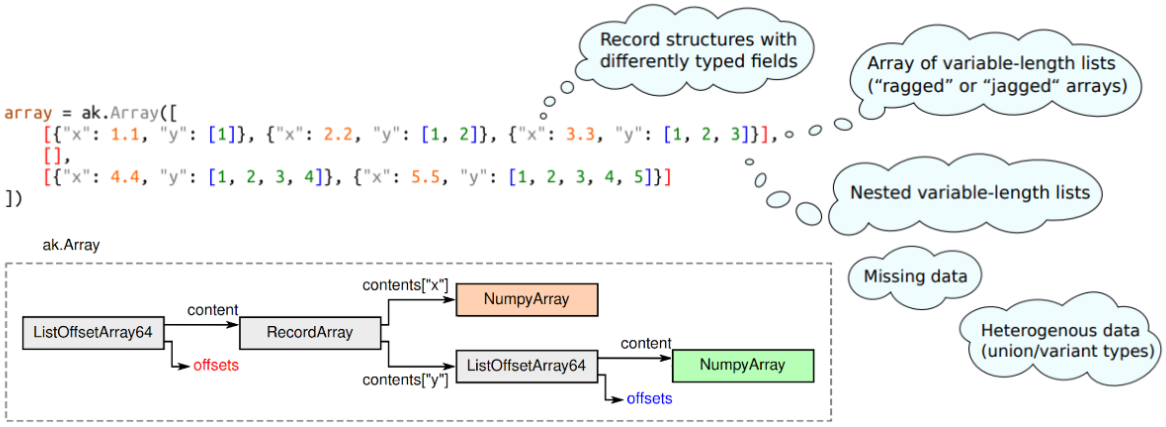


图 1. Awkward Array 的结构，其中包含嵌套的可变长度列表和记录，并以数组示例进行颜色编码。

两种语言各自的优势，从而既能拥有 Python 用户界面，又能利用 C++ 的性能和内存管理能力。高能物理领域和其他科学界已广泛投入使用 C++ 框架，并迅速将数据分析迁移到 Python。这些领域尤其希望弥合两种语言之间的差距 [8]。这引出了一个重要的问题："如何正确实现 *Python-C++* 集成？"，下文将对此展开讨论。

3. "仅头文件"方案

为解决 Awkward Arrays 与 Python 集成时的问题，现已推出一组仅包含头文件的 C++ 库 [7]。这些模板化的 C++ 库不依赖任何应用程序二进制接口。无需链接平台特定库，即可在项目编译时直接将它们包含进来。这种"仅头文件"方案不仅简化了在项目中生成 Awkward Arrays 的过程，也提高了 Awkward Arrays 的可移植性。代码精简，并未包含在 Python 中使用 Awkward Arrays 所需的全部代码。其中不包含任何对 Python 或 Python 绑定的引用。C++ 用户可以使用这些头文件创建 Awkward Arrays，然后将其复制到 Python 中，无需使用任何专用数据类型——只需原始缓冲区、字符串和整数。此方案解决了带有原生依赖项的项目打包问题。

4. LayoutBuilder

"布局"由可组合的元素构成，用于确定数组的结构。它只能构建由布局 Form 确定的特定视图。LayoutBuilder [9] 是一组在编译时使用模板的静态 C++ 类，完全以仅头文件库的形式实现。它使用仅头文件的 GrowableBuffer（图 2），该缓冲区通过智能指针实现为链表。awkward::LayoutBuilder 使用 C++ 模板专门构造 Awkward 数据结构，该结构可以填充数据，并通过 ak.from_buffers 转换为 Python Awkward Array。数据从 LayoutBuilder 输出为一组带名称的缓冲区和一个 JSON [10] Form。Form 是对 Awkward Array 的唯一描述，并返回一个 std::string，告知 Awkward Array 如何将所有内容组合起来。LayoutBuilder 是 awkward-cpp 包的一部分，与 awkward 包相互独立。这两个包都可以分别通过 pip 安装。代码没有用于将数据传递给 Python 的辅助方法，因此不同项目可以使用不同的绑定生成器。代码依赖广义 Lambda 表达式来推导参数类型

在编译时，这一功能可从 C++14 标准中获得。

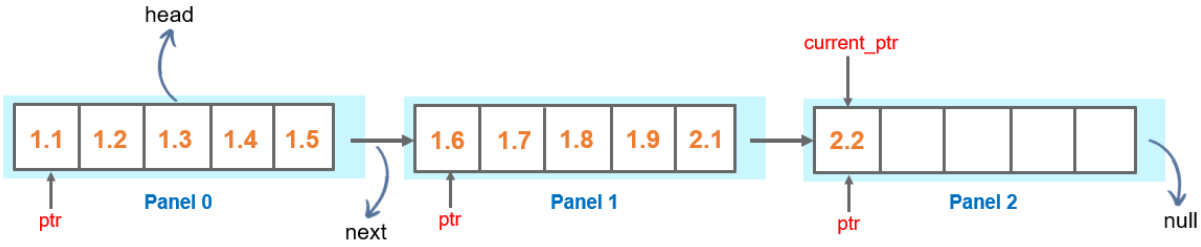


图 2. Awkward Array 的 GrowableBuffer 实现为一个链表，由多个大小为 $= 5$ 的面板组成；这些面板按需分配，即在 GrowableBuffer 空间不足时分配。

ArrayBuilder [11] 和 LayoutBuilder 都用于创建 Awkward Arrays。LayoutBuilder 与 ArrayBuilder 的主要区别在于，LayoutBuilder 可追加的数据类型是预先定义的，而 ArrayBuilder 则可以追加任意数据类型。LayoutBuilder 旨在更快地构建 Awkward Arrays。ArrayBuilder 的灵活性会带来性能限制，因为它需要推断数据类型，而 LayoutBuilder 则预先知道数据类型。

5. LayoutBuilder 的用户界面

本节将通过一个包含嵌套记录和可变长度列表的 Awkward Array 示例，介绍 LayoutBuilder 的用户界面。

5.1. LayoutBuilder 的阶段

使用 LayoutBuilder 分为三个阶段：

(i) **构造 LayoutBuilder：** 通过可变参数模板（这是一次隐式模板实例化）完成。(ii) **填充 LayoutBuilder：** 不断遍历原始指针。(iii) **将数据传递到用户分配的缓冲区：** 如有需要，用户可以将这些缓冲区传递给 Python。

5.2. 示例

列表 1 展示了 RecordBuilder 的一个示例。第一步是包含 LayoutBuilder 头文件（安装说明请参见 [9] ）。接下来，使用可变参数模板构造 RecordBuilder。RecordBuilder 的内容是异构类型容器（std::tuple），它们将其他 Builder 作为模板参数。字段名是由用户定义的非类型模板参数。目前无法将字符串用作模板参数，因为此功能仅在 C++20 及更高版本中提供。因此，为了将字段名作为模板参数传递给 RecordBuilder，必须提供一个用户定义的字段_映射，其中枚举类型的字段 ID 作为键，字段名作为值。对于多个 RecordBuilder，必须为所使用的每个 RecordBuilder 分别指定一个用户定义的映射。

之后，按照列表 1 所示，将所需数据填入 LayoutBuilder 缓冲区。为确保填充这些缓冲区时不会出错，用户可以使用 is_valid() 方法检查其有效性；如果想以安全性换取速度，也可以对每个条目调用该方法。这个示例在 Python 中转换为以下 Awkward Array：

```
[{"x": 1.1, "y": [1]}, {"x": 2.2, "y": []}, {"x": 3.3, "y": [1, 2]},]
```

```
#include "awkward/LayoutBuilder.h"

枚举 字段:std::size_t {x, y}; 用户
定义映射 字段_map({{Field::x, "
x"},{Field::y, "y"}});

// Constructing a LayoutBuilder from variadic templates!RecordBuilder<
RecordField<字段::x, NumpyBuilder<double>>,RecordField<字段::
y,ListOffsetBuilder<int64_t, NumpyBuilder<int32_t>>>> builder(fields_map);


auto& x_builder  =builder.field<字段::x>();auto&
y_builder  =builder.field<字段::y>();

// Filling the LayoutBuilder
x_builder.append(1.1);
auto& y_subbuilder = y_builder.begin_list();
y_subbuilder.append(1);
y_builder.end_list();

x_builder.append(2.2);
y_builder.begin_list();
y_builder.end_list();

x_builder.append(3.3);
y_builder.begin_list();
y_subbuilder.append(1);
y_subbuilder.append(2);
y_builder.end_list();
```

列表 1：包含嵌套记录和可变长度列表的 LayoutBuilder 示例。

我们希望由 NumPy 拥有这些数组缓冲区，这样当 Awkward Array 超出 Python 作用域时，它们就会被删除，而不是等到 LayoutBuilder 超出 C++ 作用域时才删除。因此，交接需要经过几个步骤：

(i) 获取缓冲区名称及其大小（以字节为单位）。

```
std::map<std::字符串,大小_t> 名称_字节  =
{};builder.buffer_字节(名称_字节);
```

(ii) 在 Python 中使用 np.empty(nbytes, type = np.uint8) 为这些缓冲区分配内存，并通过转换 numpy_array.ctypes.data 的输出，获取指向这些缓冲区的 void* 指针。

(iii) 让 LayoutBuilder 填充这些缓冲区。`std::map<std::string,void*> buffers;builder.to_buffers(buffers);`(iv) 最后，使用以下代码生成 JSON Form：`std::string form = builder.form();`

列表 1 中示例生成的 Form 如列表 2 所示。现在，可以使用 pybind11 [12]针对缓冲区的 py::buffer_协议，将所有内容从 C++ 传递到 Python；此外，还可以传递一个整数作为长度，以及一个字符串作为 Form。如果用户需要更改记录的格式（添加、删除、重命名字段，或更改字段类型），则无需更改 Python-C++ 接口。所有这些都包含在 C++ 模板的特化和填充过程中，而这两者都位于 C++代码中。

```
{"class": "RecordArray", "contents": { "x":
{"class": "NumpyArray", "primitive": "
float64", "form_key": "node1"}, "y": {"
class": "ListOffsetArray", "offsets": "i64", "
content": { "class": "NumpyArray", "
primitive": "int32", "form_key": "node3"},
"form_key":"node2"}" }, "form_key": "
node0"}
```

列表 2：列表 1 中示例的 Awkward Array Form。

6. 应用

仅头文件方法可应用于静态和动态项目。Awkward RDataFrame [13] 使用 C++ 仅头文件库，简化 ROOT [14]中即时（JIT）编译的过程。ak.from_rdataframe [15]函数会将所选 ROOT RDataFrame [16]列转换为原生 Awkward 数组。基于模板的仅头文件实现会根据基本数据类型 [17]构造 Form。通过模板生成所有类型，可以更轻松地从 Python 字符串动态生成 LayoutBuilder，然后使用 Cling [18]编译它。

仅头文件 LayoutBuilder 的另一个应用场景可能是 ctapipe [19]项目，该项目目前处于规划阶段。ctapipe 是一个框架，用于为切伦科夫望远镜阵列 [20]开发底层数据处理算法原型。它会对结构化（“awkward”）事件数据进行处理，开发者希望重构其实现以使用 Awkward Arrays。他们已经编写了 C++代码来遍历这种自定义文件格式，其中的数组类型在编译时已知。在这个项目中，使用 Awkward Arrays 最简单的方式是利用 LayoutBuilder 填充缓冲区，然后通过 pybind11 将它们传递给 Python。

7. 结论

本文提出的仅头文件方法促进了 Awkward Arrays 与 Python 的集成，并增强了其可移植性。一组仅头文件模板库仅使用 C 类型（整数、字符串和原始缓冲区）来构建 Awkward Arrays，并通过生成 JSON Form 将其发送到 Python。独立的 awkward 仅头文件 C++ 软件包为用户打开了在 Python 中分析数据的大门。Awkward Arrays 可以轻松集成到外部项目中，而无需链接平台特定库，也无需担心

原生依赖项。这项新进展还拓展了 Awkward Arrays 的应用范围，使其能够服务于高能物理以外的科学界。

8. 致谢

本工作得到美国国家科学基金会合作协议 OAC-1836650（IRIS-HEP）和美国国家科学基金会合作协议 PHY-2121686（US-CMS LHCOps）的支持。

引用

[1] Pivarski J、Osborne I、Ifrim I、Schreiner H、Hollands A、Biswas A、Das P、Roy Choudhury S、Smith N 和 Goyal M，2018，Awkward Array [计算机软件] *Zenodo* https://doi.org/10.5281/zenodo.4341376

[2] 2019 *EPJ Web Conf.* **214** 06005 10 1051 201921406005

[3] 2020 *Nature* **585** 357 362

Rodrigues E，Scikit-HEP 项目，DOI：. /epjconf/ Harris C R、Millman K J、van der Walt S J 等，使用 NumPy 进行数组编程 – 。DOI：10.1038/s41586-020-2649-2[4] Pivarski J、Nandi J、Lange D 和 Elmer P，2019，面向高能物理分析的列式数据处理 *EPJ Web Conf.***214**06026，DOI：10.1051/epjconf/201921406026[5] Pivarski J、Elmer P 和 Lange D，2020，Python、C++ 和 Numba 中的 Awkward Arrays *EPJ Web Conf.* **245**05023，DOI：10.1051/epjconf/202024505023[6] Pivarski J、Osborne I、Das P、Biswas A 和 Elmer P，2020，Awkward Array：类似 JSON 的数据、类似 NumPy 的惯用法 *Proc. of the 19th Python in Science Conf. (SCIPY 2020)* 78-84[7]Pivarski J，2021，Python-C++ 集成的经验教训 *20th International Workshop on Advanced Computing and Analysis Techniques in Physics Research*https://indi.to/N69ds[8]Pivarski J、Rodrigues E、Pedro K、Shadura O、Krikler B 和 Stewart G A，2022，高亮度大型强子对撞机计算评审第二阶段，通用软件项目：分析用数据科学工具 *[arXiv 2202.02194]*DOI：10.48550/arXiv.2202.02194[9]用户指南：如何在 C++ *Awkward Array Documentation* https://awkward-array.org/doc/main/user-guide/how-to-use-header-only-layoutbuilder.html[10] JSON https://www.json.org/[11] ArrayBuilder https://awkward-array.org/doc/main/reference/generated/ak.ArrayBuilder.html [12] Jakob W、Rhinelander J 和 Moldovan D，2016，pybind11——C++11 与 Python 之间的无缝互操作。https://github.com/pybind/pybind11[13] Osborne I 和 Pivarski J，2022，Awkward RDataFrame 教程 *PyHEP 2022 (virtual) Workshop* https://doi.org/10.5281/zenodo.7081586[14] Brun R、Rademakers F、Canal P、Naumann A、Couet O、Moneta L、Vassilev V、Linev S、Piparo D、GANIS G、Bellenot B、Guiraud E、Amadio G、wverkerke、Mato P、TimurP、Tadel M、wlav、Tejedor E、Blomer J、Gheata A、Hageboeck S、Roiser S、marsupial、Wunsch S、Shadura O、Bose A、CristinaCristescu、Valls X 和 Isemann R，2019，root-project/root：v6.18/02（v6-18-02） *Zenodo* https://doi.org/10.5281/zenodo.3895860[15] 来自 rdataframe https://awkward-array.org/doc/main/reference/generated/ak.from_rdataframe.html[16]Piparo D、Canal P、Guiraud E、Pla X V、Ganis G、Amadio G、Naumann A 和 Tejedor E，2018，RDataFrame：在 100 个线程上轻松进行 ROOT 并行分析 *EPJ Web Conf.* **214** 06029DOI：10.1051/epjconf/201921406029[17]Osborne I 和 Pivarski J，2022，Awkward 与 RDataFrame 之间的转换 *[arXiv 2302.09860]*DOI：10.48550/arXiv.2302.09860[18] Cling [在线]。https://root.cern.ch/cling[19] ctapipe 文档 https://ctapipe.readthedocs.io/en/latest/[20] ctapipe 天文台 https://www.cta-observatory.org/