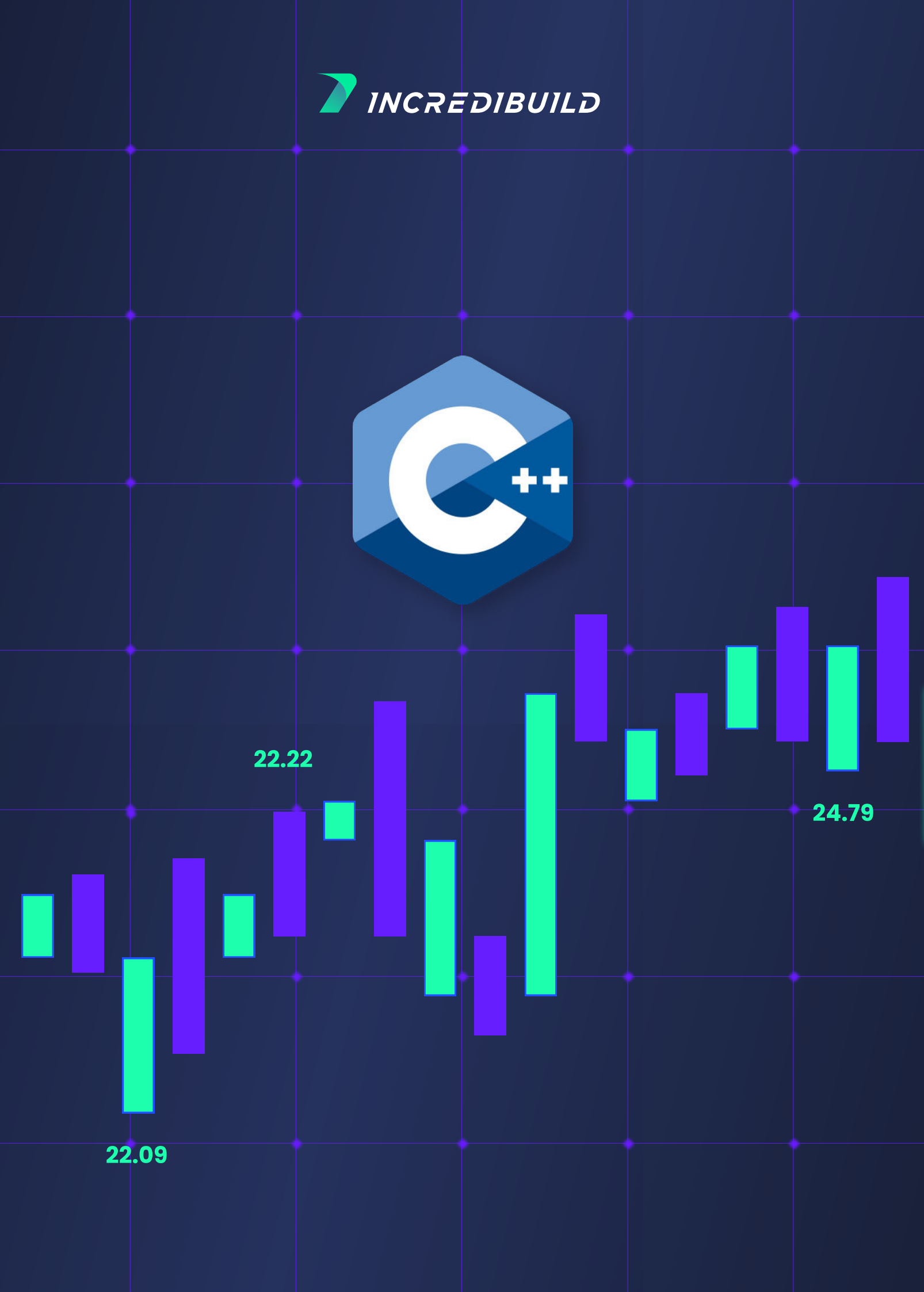
INCREDIBUILD



引言

尽管高级编程语言的引入使得软件开发更加普及，但 C++ 依然是全球技术基础设施，尤其是金融领域的基石。C++ 以其接近底层汇编语言的特性、精细的内存管理能力以及多功能性而备受青睐。

C++ 已经接近 39 岁高龄，但它依然拥有庞大的用户群体。自 1998 年以来，C++ 一直由国际标准化组织（ISO）进行标准化管理，并得到广泛的社区支持。

然而，新版本的普及并非一帆风顺。许多开发者仍然偏好使用旧版本的 C++，原因多种多样：对旧版本的熟悉、庞大的遗留系统、对迁移可能带来的兼容性问题的担忧等。

随着 C++23 版本的发布，这种状况在 C++ 社区中愈发明显。虽然新版本保持了向后兼容性，支持旧功能，但许多开发者更倾向于维持现状。

这对一些行业内的企业来说，将成为一个问题，以金融行业为例，金融服务业以其严格的监管框架和对安全合规的高要求而闻名。使用老旧的编程语言，尤其是那些被标记为潜在不安全的语言（尽管这一点存在争议），可能会影响银行和其他金融服务提供商更新其 C++ 版本的决策。本白皮书将探讨 C++23 的发布对金融服务业的影响，并讨论金融领域的开发团队是否必须进行迁移，或者他们是否可以从新版本中选择性的采纳新版本语言的优势。

你使用哪个版本的 C++？

在深入了解之前，我们先了解一下开发者偏好的 C++ 版本。根据 TIOBE 指数，C++ 在 2024 年 5 月是全球第三受欢迎的编程语言。

JetBrains 的一项调查显示，自 2017 年以来，C++ 的使用率有所增长，2023 年有 25% 的受访者使用 C++。尽管 C++ 历史悠久，但依然是开发者的坚实选择，甚至越来越受欢迎。

然而，与一些其他流行的编程语言不同，C++ 开发者在版本选择上并没有达成共识。许多开发者已经转向最新版本（C++20），但这部分人群仍属少数。根据 JetBrains 的数据，只有 23% 的受访者迁移到了 C++20，而大多数仍在使用 C++17，31% 仍在使用 C++11。

更重要的是，近一半的受访者没有计划迁移到更新的语言标准。这种现象引发了一个问题：为什么这么多的组织和开发者没有积极推动新标准的采用？

为何坚持旧标准？

最主要的原因是习惯。对于许多开发者来说，接受新标准意味着重新学习、暂停关键工作以及将现有代码迁移到新标准，这些都是不小的挑战。

此外，C++ 在向后兼容性方面做得很好，允许旧版本使用新标准中的功能。

资源有限

另一个重要原因是实际操作中的资源限制。无论是开发人员的时间、产品停机时间还是财务成本，组织和开发团队可能没有足够的资源将整个代码库迁移到最新版本。

不升级的代价

当我们讨论迁移、更新和新标准时，最关心的是这些变化是否会给团队带来负面影响。许多新标准在发布时对各种功能的支持不完整，这可能会在生产环境中引发问题。一个不完全工作的编译器可能会引入长期稳定运行的代码中的 bug，而未完全实现的新功能可能会在实时应用程序中造成不利影响。

不升级的真正影响

新标准采用率的差异，以及像 C++98 这样老旧版本的普遍存在，引发了一个问题：是否真的有必要升级？如果可以在不牺牲代码库或应用程序质量的情况下进行选择，那么代价是什么？

这种观点虽然有一定道理，但忽略了迁移和采用更新标准带来的一些积极影响。一些团队可能不觉得有必要为了获得递增的改进而迁移代码库。

然而，即使是递增的改进，也可能为开发者带来显著的提升，并使应用程序总体上变得更好。以下是升级到新标准的一些好处：

- 利用新语法或库选项扩展团队的能力，带来更智能的解决方案、更好的代码，甚至在构建和运行时都提高性能。
- 使用最新版本附带的标准库和功能，而不是依赖外部库，使代码更易于维护，也更容易让新成员加入。
- 保持代码的现代性。尽管有些抵制，但开发者喜欢进步，他们喜欢感觉自己正在与项目一起前进，而不仅仅是维护旧代码。
- 落后可能不总是意味着停留在操作系统或编译器版本上，但它确实足够频繁。这会带来从兼容性到在新版本中修补的安全问题等不必要的风险。

技术债务：避免落后的陷阱

当讨论迁移（或不迁移）到新标准时，一个经常被忽视的问题是，坚持现状可能会在未来引发问题。技术债务的积累可能不是公开或明显的过程，但当一个小问题变成一个重大危机时，它可能会成为一个关键问题。

虽然 C++ 不像一些软件那样被故意弃用或使过时，但它并没有使用先前标准的硬障碍。即便如此，社区和标准委员会将积极支持该语言的最新版本。这意味着不升级可能会导致一些需要变通的兼容性问题，不再支持的功能等。

没有明确的答案来回答是否升级的问题。最终，不同的组织和开发者可能有自己的哲学，迁移将取决于各种因素。即便如此，目标应该是最终转向更近期的标准。

C++ 与金融——自然匹配

在我们探讨最新的 C++ 标准及其对开发者的意义之前，让我们深入了解金融和金融服务提供商如何适应 C++ 生态系统。自 20 世纪 90 年代中期以来，C++ 一直是金融行业开发的关键部分。金融建模的复杂性意味着更线性的语言越来越不适合处理负载，而 C++ 提供了更全面的对象导向方法。

C++ 被采用的第二个关键因素是其在时间敏感和资源密集型任务中的卓越性能。随着金融更全球化模型的出现，数据需要在接收时立即处理，这意味着 C++ 是一个显而易见的明智选择。它更好地管理内存的能力也对许多金融模型的重型处理需求至关重要。

如今，即使有其他提供更快开发选项的语言，C++ 的运行时效率意味着它仍然被广泛使用。不同之处在于，像 Python、R 和 MatLab 这样的解释性语言每次执行代码时都需要处理，而 C++ 可以避免这种情况。更重要的是，这些工具并不是相互排斥的。C++ 可以处理复杂和资源密集型的任务，而像 R 和 MatLab 这样的高级语言可以更好地实现 C++ 原生不支持的可视化和建模工具。

C++ 在金融服务中的关键用例

高频交易

高频交易（HFT）依赖于交易系统对市场变化、新数据的几乎即时反应，并且可以以极低的延迟执行。这就是 C++ 大放异彩的地方——提供运行时的高性能，同时与硬件进行尽可能快速和低级别的交互。

C++ 算法旨在与硬件加速器交互，这些加速器的构建旨在提供最低的延迟（当交易需要在微秒内输入和执行时，这是一个关键需求）。

风险管理

在这里，C++ 的对象导向特性使其成为必须考虑各种外部数据源并在实时进行计算的复杂模型的理想选择。由于其便携和灵活的特性，C++ 程序可以快速解析来自各种输入的数据，而不影响输出时间。

更重要的是，C++ 可以很好地与提供更强大图形界面支持、数据可视化和其他更高级统计工具的语言（如 Python、R 和 MatLab）一起工作。

定价模型

定价引擎必须快速分析各种来源的数据，以提供合乎逻辑并反映现实的金融工具价格。C++ 的对象导向特性在这方面很有帮助。

- Black-Scholes 模型：这是一个常用的期权合约定价模型。这个模型需要大量的数据集和高度复杂的数学计算，C++ 非常适合快速解决。
- 蒙特卡洛模型：这个模型用于更奇异的期权，需要完成多次模拟以达成共识价格。C++ 可以快速运行这些模拟，延迟和停机时间很低。
- 利率曲线构建：这个模型构建了一个显示随时间变化的利率相关性的曲线，这对于定价各种工具至关重要。
- 收益率曲线引导：这是一个前瞻性模型，从收益率曲线中提取隐含的远期利率。C++ 的高性能和准确性对此至关重要。

.

量化分析

C++ 与其他语言一起工作，产生复杂的模型，帮助公司构建先进的交易策略。一方面，C++ 并不是为统计分析而内置的——这就是 MatLab 和 Python 等语言的用武之地。另一方面，这些语言缺乏 C++ 所拥有的低级效率和性能能力，因为它们的代码每次运行时都必须被解释。

在这种情况下，C++ 用于运行庞大的数据集和实际计算，而其他语言执行数据的统计和可视化。

交易成本分析

这种类型的分析测量执行交易的成本，这在高频交易和算法交易的世界中变得越来越重要。关键用例是确定不同策略的有效性和投资回报率，基于它们的成本。C++ 的内存管理能力和其快速管理大型数据集的能力使其非常适合这项任务。

C++ 标准与金融——复杂的关系

金融服务行业不断面临二元性——迫切需要创新，拥抱现代技术并突破边界，但同时存在着严格的监管和合规框架，使其难以快速有效地创新。

新的变化伴随着未知数——无论是像创新技术这样的重大转变，还是像现有软件的新版本这样的小变化。新的软件标准可能会引入出色的功能，这些功能可能会引起重大的安全风险，因此对金融服务采用新的软件可以会给资深带来风险。

特别是当讨论代码和现有应用程序的时，问题就变成了是否值得迁移代码。答案大多不出所料，尽管新代码可以使用新标准，但迁移旧代码库带来了太多的陷阱和问题，不值得这样做。已经符合严格安全和监管框架的代码可能会突然变得不那么符合，因为新功能没有完全实现或测试。

此外，惯性在抵制接受新标准中发挥了作用。需要教授和学习新模块、新库等意味着浪费时间。此外，许多开发者并不觉得有必要接受不会增加显著改进的新标准。在金融领域，这更是如此，因为一些模型和日常执行的计算所需的复杂代码库可能会因为最小的意外变化而中断。

即便如此，新代码不太可能有这个问题，因此组织更有可能在创建新代码库时接受新标准。如果现有工具和模型中断的风险很小，开发者可以更自由地尝试新事物，采用更高性能或简化他们代码的工具。

C++23 为金融服务带来了什么?

C++23 不仅仅是开发中的巨变，更是对以前标准的提炼和逐步改进。即便如此，有一些功能肯定会吸引行业内开发人员的注意。

协程

虽然 C++20 有一个实现协程的框架，但没有具体的方法来部署它们。这在 C++23 中通过引入 std::generator 得到了解决。这是一个新的库，允许开发者通过从它返回的协程中恢复来生成元素序列。

这将简化生成复杂数据序列的过程，并且可以比在以前的标准中更容易地部署。这不是一个巨大的变化，但它确实简化了在更大的代码库中使用协程的过程。

范 围

C++23 中范围的最新功能是增加了视图，允许非拥有访问数据。用户现在可以直接与数据本身交互，而不是访问数据容器本身。更重要的是，C++23 增加了许多在 C++20 中没有插入的函数。

Range-v3 增加了显著的查看能力，允许访问数据和各种不会显著影响性能和代码复杂性的视图。最后，zip 范围适配器的增加允许在范围创建方面具有更大的灵活性——包括替换几个主要因为缺少像 zip 这样的功能而存在的算法。

工作体验的提升

C++23 带来的改进远不止于此，但相较于之前的版本，这次标准的更新更多是渐进式的。许多新增的特性旨在提升编程的便捷性，同时弥补了 C++20 及其之前版本中的一些不足。需要注意的是，部分新特性可能与某些编译器不完全兼容。

- **智能指针:** 新版智能指针特别适用于与 C 语言 API 紧密集成的代码。它们简化了程序员常用的一种复杂抽象——即“指针的指针”。开发者现在可以直接使用 std::out_ptr 和 std::in_out_ptr 来实现这一功能，这样做不仅代码更清晰，性能也更优。这些智能指针在它们被创建的表达式执行完毕后会被自动销毁，从而避免了悬空指针的问题。

- **Std::expected**: 这一特性允许程序员存储一个预期值或一个意外值（也就是说，这个对象永远不会无值）。它通过隐式声明可能发生的意外值，使得异常和错误的处理更为简洁，减少了对预期异常和潜在错误的显式处理。

安全性和可靠性的提升

对于金融服务行业的开发者来说，C++23 中增强安全性的特性尤其吸引人。这些特性包括：

- **类型安全性的改进：** C++23 引入的新语言特性和标准库改进加强了类型安全性，减少了运行时错误的风险。这意味着在应用程序运行时出现意外错误和暴露漏洞的可能性更低，这些漏洞在编码阶段可能并不明显，但在应用程序上线后可能会显现出来。
- **代码安全性的提升：** 标准库和语言特性的更新有助于开发者编写更安全的代码，这对于处理敏感数据的金融应用至关重要。

逐步接受变化

最终，采用新标准并没有绝对意义上是或者否。事实上，最好的实施方式是选择最佳功能，同时保留已经使用并支持了一段时间的以前标准的稳定性。

在金融服务行业中，尤其如此。该领域以对变化尤其谨慎，无论是因为安全问题、遗留代码和应用程序、现有功能、或是性能问题等，引入新变量可能会引起大多数开发者抵触。即便如此，变化并不总是坏事。它确实需要一个微妙的触感，以及对如何引入新标准、新库、新功能等可能影响现有流程的深刻理解。

当谈到 C++ 新版本时，好处是新标准仍然向后兼容，允许开发人员限制风险敞口，并继续有效地开发应用程序，同时避免使用的语言过于落后以及完全过时的代码库。开发团队的目标应该始终是接受新标准，并保持与新趋势的同步。虽然它们可能会带来一些挑战和风险，但最终它们更有可能成为提高代码质量的有效方式，在性能至关重要的行业中，毫秒可以意味着百万美元利润和百万美元损失，使用新标准可以是企业更接近成功。

关于 Incredibuild

Incredibuild 是全球领先的开发加速平台。我们的“虚拟分布式处理”和 Build Cache 专利技术帮助开发团队在本地和云中更快地构建并降低开发成本，以便可以实现更多迭代、更快修复错误，并构建出色的产品。

除此之外，Incredibuild 还帮助开发团队充分利用现有硬件、优化云资源，降本增效。

Incredibuild 不需要更改现有的工具或流程；它是一个轻量级、无代码的解决方案，大多数团队可以在几小时内启动。Incredibuild 旨在提升分布式开发团队的开发效率，即使是那些使用带宽有限的家庭网络办公的开发人员也可以体验这种效率的提升。