# Apache Nifi深入

Apache Nifi团队 – dev@nifi.apache.org

## 目录

简介
存储库
FlowFile存储库
系统故障对交易的影响
更深的视图：内存和磁盘上的流files
内容存储库
更深的视图：内容主张
出处存储库
更深的视图：出处日志文件
一般存储库注释
多个物理存储点
最佳实践
流file的生活
WebCrawler流
数据入口
通过参考
扩展路由用例
funnels
复制写
写入用例上的扩展副本
更新属性
典型的用例注释
数据出口
更深的视图：检查点后删除
关联不同的数据
闭幕词

## 简介

此高级文档旨在深入了解NIFI的实施和设计决策。它假设读者已经阅读了足够多的其他文档，以了解NIFI的基础知识。

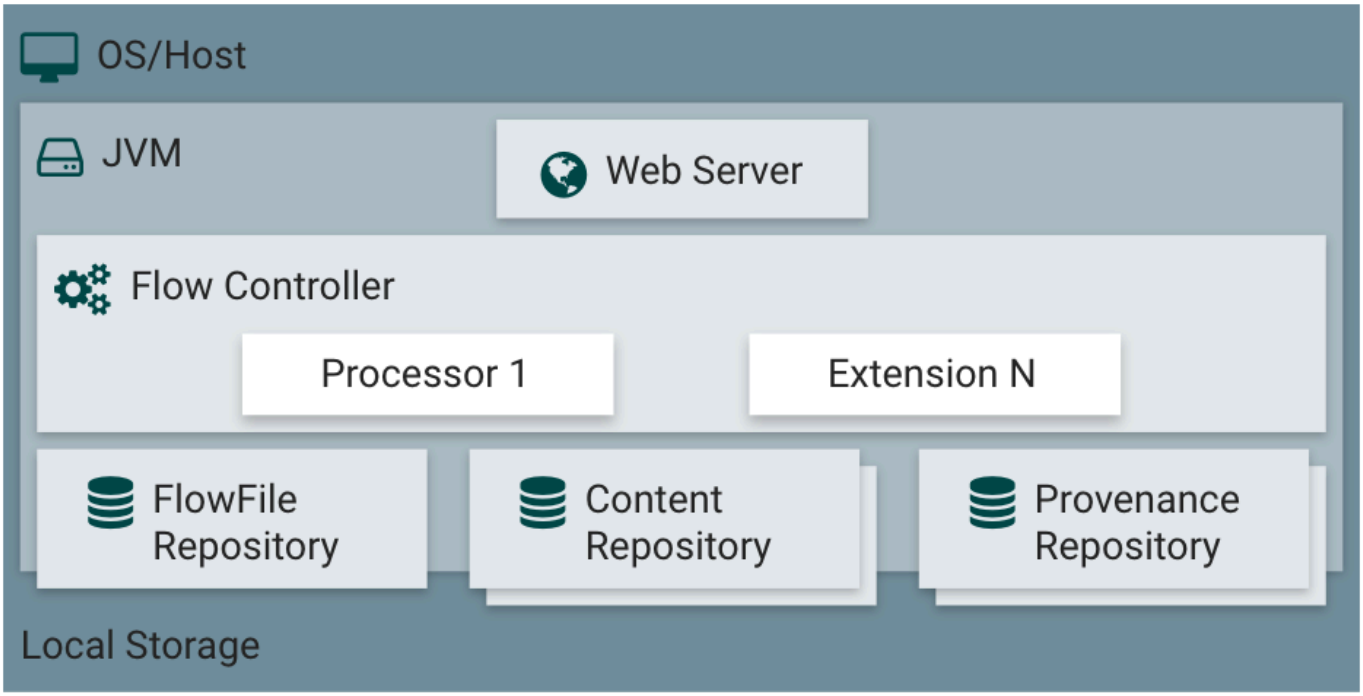
FlowFiles是NIFI及其基于流程设计的核心。 FlowFile是一个数据记录，它由指向其内容（有效载荷）的指针和支持内容的属性，与一个或多个出处事件相关联。属性是钥匙/值对，充当流file的元数据，例如FlowFile文件名。内容是实际数据或文件的有效载荷。出处是流file发生的事情的记录。这些部分中的每个部分都有其自己的存储库（存储库）。

存储库的一个关键方面是不变性。内容存储库中的内容和流文件存储库中的数据是不可变的。当更改FlowFile的属性时，将在内存中创建属性的新副本，然后在磁盘上持续存在。当给定流file更改内容时，其原始内容将被读取，通过转换流进行流，并写入新流。然后将FlowFile的内容指针更新为磁盘上的新位置。结果，可以说FlowFile内容存储的默认方法是不可变的版本的内容存储。这一点的好处包括：典型的处理，自然重播能力，利用OS缓存，降低随机读取/写入性能命中所需的典型复杂图所需的存储空间，大大降低了存储空间，并且易于推理。先前的修订根据“ Nifi.properties”文件中设置的归档属性保存，并在NIFI System Administrator的指南中概述。

## 存储库

NIFI使用了三个存储库。每个都存在于OS/主机的文件系统中，并提供特定的功能。为了充分了解流files以及基础系统如何使用它们，重要的是要了解这些存储库。这三个存储库都是NIFI用来持续数据的本地存储的目录。

- 流文件存储库包含流动中所有当前流file的元数据。
- 内容存储库保存当前和过去的流文件的内容。
- 出处存储库持有FlowFiles的历史。



## FlowFile存储库

系统积极处理的流纸在JVM内存中的哈希地图中保存（在更深视图中更多地：在内存中和磁盘中的流file）。这使得处理它们非常有效，但是由于多种原因，例如功率损失，内核恐慌，系统升级和维护周期，因此需要一种二级机制，以在过程重新启动过程中提供数据耐用性。 FlowFile存储库是系统中当前存在的每个流件的元数据的“写入日志”（或数据记录）。该流file元数据包括与流file关联的所有属性，指向流file的实际内容（内容存储库中）和流flow的状态，例如流file属于哪个连接/排队属于。此写入日志提供了nifi提供的NIFI提供了需要重新开始的弹性和意外的系统故障。

流文件存储库充当NIFI的写入日志，因此，当流量文件流过系统时，每个更改都会在流汇存储库中记录在其作为交易单位工作单元之前。这允许系统在处理一块数据时确切知道节点在什么步骤上。如果该节点在处理数据时会降低，则可以轻松地从重新启动时恢复的位置恢复（更深入地在交易中的系统故障生效）。日志中流件的格式是沿途发生的一系列三角洲（或更改）。 NIFI通过还原流file的“快照”（在存储库被检查时创建），然后重播这些三角洲。

该快照是由系统自动自动拍摄的，该系统为每个流file创建一个新的快照。该系统通过在哈希地图中序列化每个流文件并使用文件名“ .partial”将其写入磁盘来计算新的基本检查点。随着检查点的进行，新的FlowFile基线被写入“ .Partial”文件。检查点完成后，删除了旧的“快照”文件，并且“ .partial”文件更名为“快照”。

系统检查点之间的时期可在“ Nifi.properties”文件（在NIFI System Administrator's Guide中记录）中进行配置。默认值是两分钟的间隔。

### 系统故障对交易的影响

NIFI通过记录当时在各自的流file回购中的每个节点上发生的事情来保护硬件和系统故障。如上所述，FlowFile Repo是Nifi的写入日志。当节点返回在线时，它可以通过首先检查“快照”和“ .partial”文件来恢复其状态。该节点要么接受“快照”，然后删除“ .partial”（如果存在），或者如果不存在“快照”文件，则将“ .partial”文件重命名为“快照”。

如果节点在掉落时写入内容的中间，则没有任何损坏，这要归功于Write（如下所述）和不成度（如下所述）范式的副本。由于FlowFile交易从未修改原始内容（由内容指针指向），因此原始内容是安全的。当Nifi倒下时，对更改的写入索赔是孤儿，然后通过背景垃圾收集进行清理。这为最后一个已知的稳定状态提供了“回滚”。

然后，节点从流file恢复其状态。有关该过程的更深入，分步的解释，请参见NIFI的写入日志实现。

就工作的交易单位而言，这种设置使NIFI在面对逆境时具有非常韧性，以确保即使NIFI突然被杀死，它也可以备份而不会丢失任何数据。

### 更深的视图：内存和磁盘上的流files

术语“ FlowFile”有点错误。这将导致人们相信每个流纸都对应于磁盘上的一个文件，但这不是事实。有两个主要位置存在流file属性，上面解释的写入日志以及工作记忆中的哈希映射。该哈希映射具有对流动中积极使用的所有流file的引用。该地图引用的对象与处理器使用并在连接队列中保存的对象相同。由于流file对象保存在内存中，因此要使处理器获取流file必须完成的所有操作是要求过程中的流程从队列中抓住它。

当FlowFile发生更改时，三角洲将写入写入日志，并相应地修改内存中的对象。这使系统能够快速与流纸件合作，同时还可以跟踪会议进行时发生的事情以及将会发生的事情。这提供了一个非常健壮且耐用的系统。

还有“交换”流files的概念。当连接队列中的流files数量超过“ nifi.queue.swap.threshold”属性中设置的值时，就会发生这种情况。连接队列中优先级最低的FlowFile被序列化，并以10,000批次的“交换文件”写入磁盘。然后从上面提到的哈希地图中删除这些FlowFiles，并且连接队列负责确定何时将文件交换为存储器。交换流文件时，通知流file回购，并保留交换文件的列表。检查点检查点时，快照包含一个用于交换文件的部分。当交换文件交换回去时，流files将添加回哈希地图。这种交换技术就像大多数操作系统执行的交换一样，它使NIFI可以非常快速地访问正在积极处理的流文件，同时仍允许流量中存在数百万个流载液，而不会耗尽系统的内存。

## 内容存储库

内容存储库只是在本地存储中的一个地方，其中所有流文件的内容都存在，通常是三个存储库中最大的。如“介绍性”部分中所述，此存储库利用不变性和抄写范式来最大化速度和线程安全。影响内容存储库的核心设计决定是将FlowFile的内容保存在磁盘上，并且仅在需要时将其读为JVM内存。这使NIFI可以处理微型和庞大的对象，而无需生产者和消费者处理器将完整的对象保存在内存中。结果，诸如分裂，聚合和转换非常大的对象之类的动作很容易做到而不会损害记忆。

以同样的方式，JVM Heap具有垃圾收集过程，可以在需要空间时回收无法实现的对象，在NIFI中存在一个专用线程，以分析未使用内容的内容回购（“更深视图：检查点后的删除”部分中的更多信息”部分。将流file的内容确定为不再使用的内容后，它将被删除或存档。如果在“ nifi.properties”中启用了归档，则flowfile的内容将存在于内容储备金中，直到它老化（在一定的时间后删除）或由于内容回购占用太多空间而删除。归档和/或删除的条件在'nifi.properties'文件（“ nifi.content.repository.archive.max.max.retention.period”，“ nifi.content.repository.maxive.max.max.usage.usage.usage.usage.centage.percentage”）中，并在NIFI系统的指南中列出。有关内容删除的更多信息，请参阅“数据出口”部分。

### 更深的视图：内容主张

通常，在谈论流file时，对其内容的引用可以简单地称为内容的“指针”。但是，流文件内容参考的基本实现具有多个复杂性。内容存储库由磁盘上的文件集合组成。这些文件被纳入容器和部分。部分是容器的子目录。可以将容器视为内容存储库的“根目录”。但是，内容存储库可以由许多容器组成。这样做是为了使NIFI可以并行利用多个物理分区。”然后，NIFI可以并行读取并写入所有这些磁盘，以实现单个节点上磁盘吞吐量的数百兆字节甚至千兆字节的数据速率。 “资源声明”是Java对象，指向磁盘上的特定文件（这是通过跟踪文件ID，文件为中的部分以及该部分的一部分来完成的）。

为了跟踪FlowFile的内容，FlowFile具有“内容声明”对象。此内容索赔的引用是对包含内容的资源声明，文件中内容的偏移以及内容的长度。要访问内容，内容存储库使用资源索赔属性在磁盘上使用特定文件向下钻取，然后寻求资源声明指定的偏移，然后从文件中流式传输内容。

完成了这一层的抽象（资源声明），以使每个流file内容的磁盘上都没有文件。不变性的概念是可能的关键。由于内容一旦编写就永远不会更改（“写入写”是为了进行更改），因此，如果流file的内容更改，则不会将内存或移动数据碎片。通过利用磁盘上的一个文件保存许多流文件的内容，NIFI能够提供更好的吞吐量，通常接近磁盘提供的最大数据速率。

## 出处存储库

出处存储库是存储每个流纸的历史记录的地方。该历史记录用于提供每个数据的数据谱系（也称为监护链）。每次为FlowFile发生事件（创建，叉式，克隆，修改等）时，都会创建一个新的出处事件。该出处事件是流file的快照，因为它看起来并适合当时存在的流程。创建出处事件时，它将所有流file的属性和指针复制到流file的内容，并将其与流file的状态（例如与其他出处事件的关系）汇总到出处回购中的一个位置。除了已过期的数据外，此快照不会改变。出处存储库在完成后的一段时间内保存所有这些出处事件，如“ Nifi.properties”文件中的指定。

由于所有流file属性和内容指针都保存在出处存储库中，因此数据流管理器不仅能够看到该数据的谱系或处理历史记录，而且还可以随后查看数据本身，甚至可以从流量中的任何点重复数据。一个常见的用例是当特定的下游系统声称尚未收到数据时。数据谱系可以准确显示何时将数据传递到下游系统，数据的外观，文件名以及发送到数据的URL，或者可以确认数据确实从未发送过。无论哪种情况，都可以通过单击按钮（或访问适当的HTTP API端点）来重播发送事件，以便将数据仅重新发送到该特定的下游系统。或者，如果数据未正确处理（也许首先要进行某些数据操作），则可以固定流程，然后可以将数据重播到新流中，以便正确处理数据。

不过，请记住，由于出处没有复制内容存储库中的内容，而只是将FlowFile的指针复制到内容中，因此可以在删除其参考的出处事件之前删除内容。这意味着用户将不再能够看到内容或以后重播流file。但是，用户仍然能够查看FlowFile的血统并了解数据发生了什么。例如，即使数据本身将无法访问，用户仍然能够看到数据的唯一标识符，其文件名（如果适用），收到的何时收到，收到的位置，如何操纵，发送的位置等等。此外，由于提供了流file的属性，因此DataFlow Manager能够理解为什么以其方式处理数据，从而提供了一个重要的工具来理解和调试数据流。

由于出处事件是流file的快照，因为它存在于当前流程中，因此流量的变化可能会影响以后重播出处事件的能力。例如，如果从流中删除了连接，则无法从流程中重播数据，因为现在无处可在任何地方加入处理数据以进行处理。

要查看出处存储库背后的设计决策，请查看持续的出处存储库设计。

### 更深的视图：出处日志文件

每个出处事件都有两个地图，一个用于事件前的属性，一个用于更新的属性值。通常，出处事件不会存储事件发射时存在的更新值，而是将其属性值储存时的属性值。这些事件将被缓存并保存，直到会话进行为止，并且一旦进行了会话，则在会话进行时会发出与FlowFile关联的属性。此规则的例外是“发送”事件，在这种情况下，该事件包含发出事件时存在的属性。这样做是因为如果属性本身也发送了，那么准确地说明发送了哪些信息很重要。

随着NIFI的运行，有一个由16个出处日志文件组成的滚动组。由于发出出处事件，它们被写入16个文件之一（有多个文件以增加吞吐量）。日志文件定期滚动（默认时间范围为每30秒）。这意味着新创建的出处事件开始写入一个由16个日志文件组成的新组，并处理原始文件以进行长期存储。首先，将日志滚动到一个文件中。然后，该文件被选择压缩（由“nifi.provenance.repository.compress.on.rollover”属性确定）。最后，使用Lucene索引这些事件并可用于查询。这种批次索引的方法意味着出处事件无法立即进行查询，但是作为回报，这种批准的方法大大提高了性能，因为进行交易和索引是非常昂贵的任务。

单独的线程处理出处日志的删除。管理员可以设置的两个条件可以控制出处日志的删除，这是它可以占用的最大磁盘空间量以及日志的最大保留持续时间。该线程按最后一个修改日期对存储库进行排序，并在超过条件之一时删除最古老的文件。

出处存储库是一个卢克（Lucene）指数，分为多个碎片。这是出于多种原因这样做的。首先，Lucene对文档标识符使用32位整数，因此Lucene不碎片支持的最大文档数量是有限的。其次，如果我们知道每个碎片的时间范围，则可以轻松使用多个线程进行搜索。同样，这种碎片还可以更有效地删除。 Nifi等待，直到将碎片中的所有事件计划用于删除，然后从磁盘中删除整个碎片。这使得我们无需更新Lucene索引时。

## 一般存储库注释

### 多个物理存储点

对于出处和内容存储库，可以选择将信息跨越多个物理分区。如果管理员想在多个磁盘上结合阅读和写作，他们将这样做。存储库（内容或出处）仍然是一个逻辑商店，但是系统会自动在多个卷/分区中划分。目录在“ nifi.properties”文件中指定。

### 最佳实践

它被认为是尽可能少次分析流file的内容的最佳实践，而是将关键信息从内容物中提取到流file的属性中。然后从flowfile属性中读取/写入信息。一个例子是Extracttext处理器，该处理器从FlowFile内容中提取文本并将其作为属性，以便其他处理器可以利用它。这比不断处理流汇的整个内容提供了更好的性能，因为属性保持在记忆中，并且更新流file存储库的速度要比更新内容存储库的速度要快得多，因为每个存储库中存储的数据量。

## 流file的生活

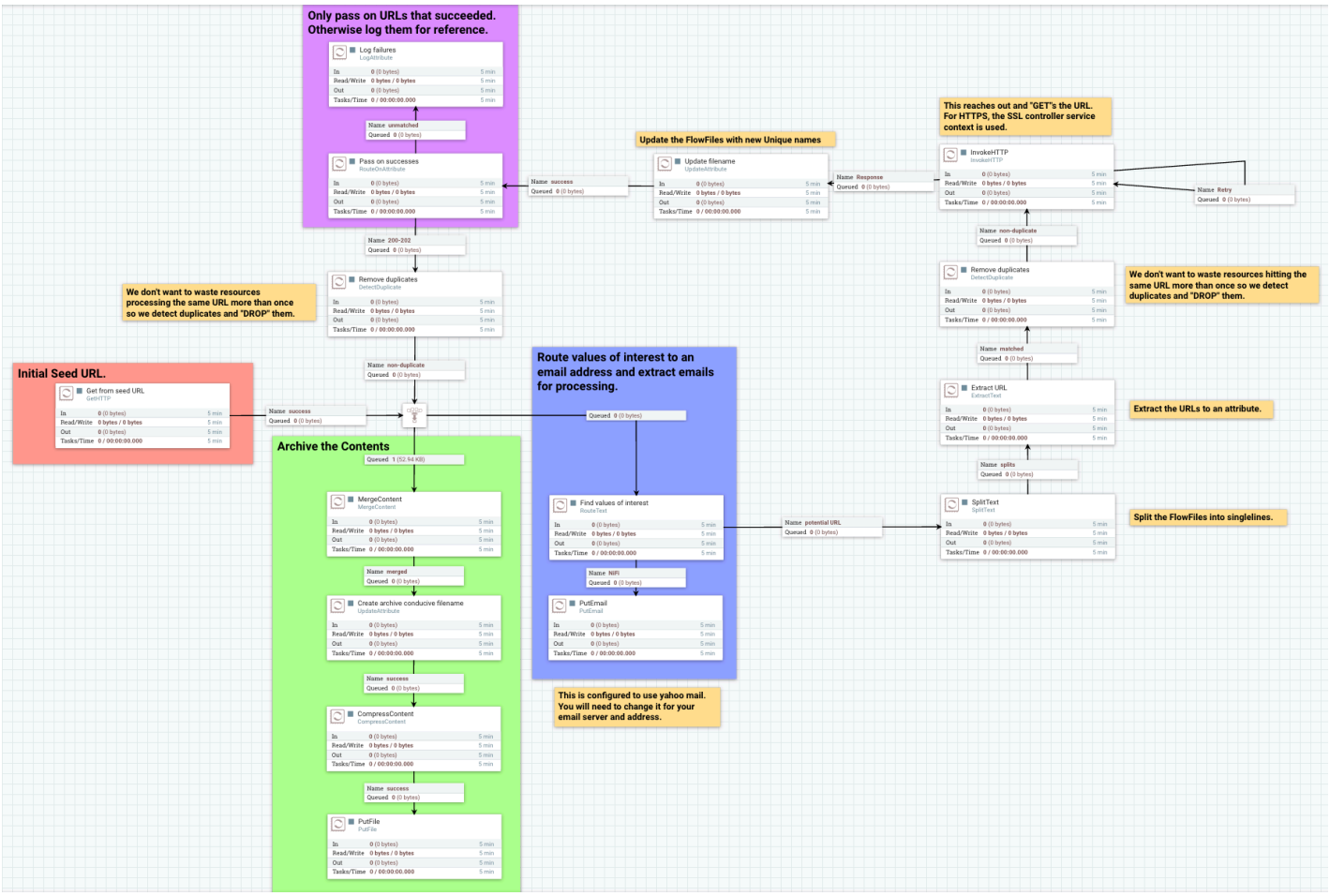
为了更好地理解存储库如何相互作用，NIFI的潜在功能以及流file的寿命；下一节将包括真实流中不同点的流file的示例。这里可用称为“ WebCrawler”的流量：webcrawler.json。

在高级别上，该流程与IndokeHTTP处理器中配置的种子URL接触，称为从种子URL中获得GET，然后使用RuteText处理器分析响应以查找关键字的实例（在这种情况下，在这种情况下为“ NIFI”），并击中了潜在的URL。然后使用原始种子网页中的url执行HTTP GET请求。响应是根据状态代码属性进行路由的，并且仅将200-202状态代码路由回到原始的RuteText处理器进行分析。

该流程还检测到重复的URL并防止处理它们，在找到关键字时向用户发送电子邮件，记录所有成功的HTTP请求，并捆绑成功的请求，以在磁盘上压缩和存档。

要使用此流程，您需要配置几个选项。首先，必须使用默认属性添加DistributeMapcacheserver Controller服务。为了获取电子邮件，必须使用您的电子邮件凭据配置Putemail处理器。最后，要使用HTTPS，必须使用适当的密钥和信任商店配置标准Slcontextservice。请记住，必须使用适当的证书机构配置信托店才能为网站工作。下面的命令是使用“ keytool”命令将默认Java cas添加到一个名为MyTrustStore的信托店的示例：KEYTOOL -IMPORTKEYSTORE -SRCKEYSTORE/library/java/javavava/javavirtualmachines/jdk/jdk/jdk/jdents/jdecontents/home/home/jre/jre/jre/lib/lib/cacerts -cacerts -desteystekeystore mytrusteTore mytrystore mytrstore mytrstore mytrtorstore mytrstore mytrtorstore mytrtorstore

# WebCrawler流



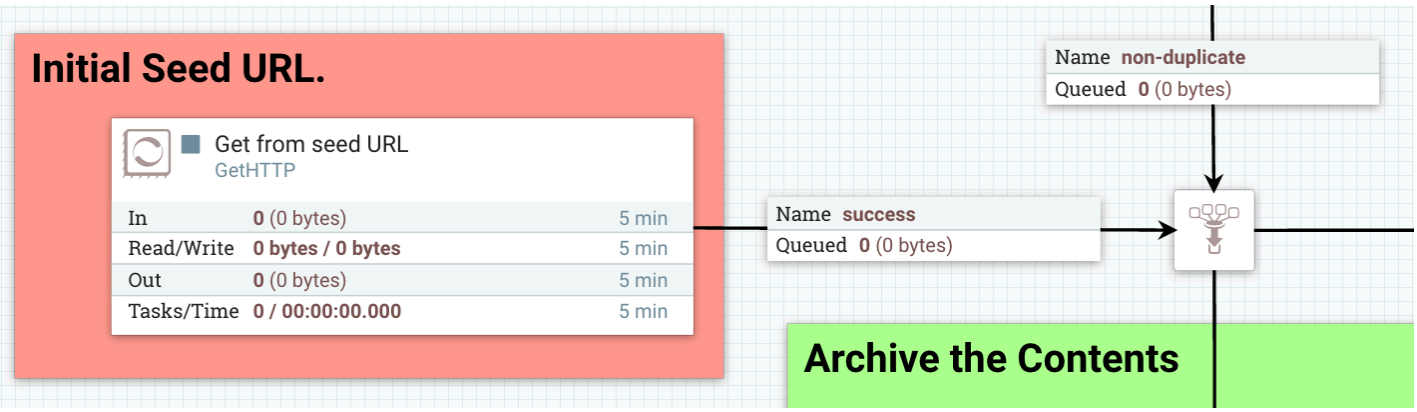

由于Web爬网的随机性质，在InvokeHTTP处理器上出现诸如“连接时间”之类的消息的公告并不少见。

## 数据入口

当生产者处理器调用“ processSession.create（）”，然后对Provenancereporter进行适当的呼叫时，在系统中创建流file。“ processSession.Create（）”呼叫创建一个具有一些核心属性（标准过程会话的文件名，路径和UUID）的空flow文件，但没有对父母的任何内容或谱系（将创建方法重载超载以允许对父flowfiles的参数）。然后，生产者处理器将内容和属性添加到FlowFile中。

Provenancereproter用于发射流file的出处事件。如果文件是由NIFI从外部实体未收到的数据创建的，则应发出“创建”事件。相反，如果从外部来源收到的数据创建数据，则应发出“接收”事件。出处事件分别使用“ provenancere.create（）”和“ provenancere.receive（）”进行。

在我们的WebCrawler流中，称为“从种子URL”的InvokeHTTP处理器使用“ ProcessSession.Create（）”创建初始FlowFile，并使用“ ProvenanceReporter.Receive（）”记录了数据的接收。此方法调用还提供了收到数据的URL，传输数据的时间以及添加到流汇中的任何流file属性。例如，可以将HTTP标头添加为FlowFile属性。



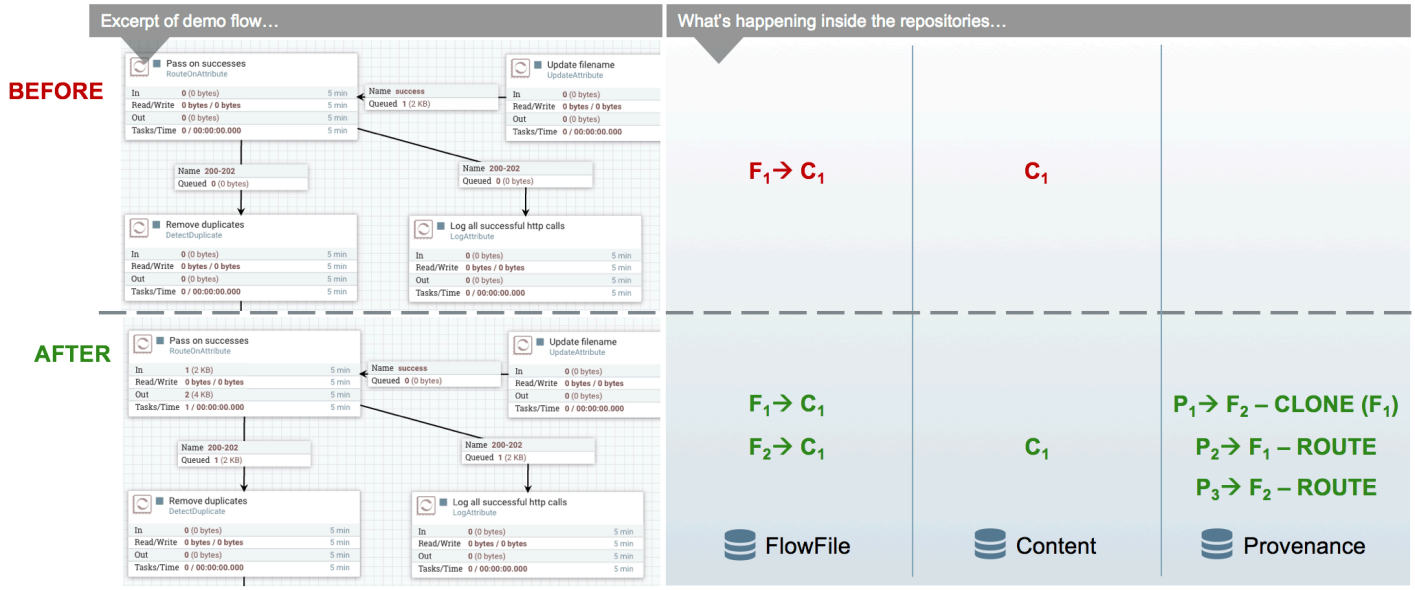

## 通过参考

基于流程的编程的一个重要方面是黑匣子之间资源约束关系的想法。在Nifi中，这些分别是队列和处理器。流量文件仅通过传递对流file的引用（类似于EIP中的“索赔检查”模式），从一个处理器到另一个处理器。

在WebCrawler流中，InvokeHTTP处理器通过HTTP获取请求与URL联系，并根据HTTP服务器的响应是什么，将状态代码属性添加到FlowFile中。更新FlowFile的文件名（在InvokeHTTP之后的UpdateAttribute处理器中）后，有一个RouteOnattribute处理器，该处理器将成功的状态代码属性汇总到两个不同的处理器。 RouteonAttribute处理器“删除”了无与伦比的那些（请参阅数据出口部分），因为它配置为自动终止与任何不匹配任何路由规则的数据。进入RouteOnattribute处理器，有一个流file（F1），其中包含状态代码属性并指向内容（C1）。有一个出处事件指向C1，其中包括F1的快照，但被省略以更好地关注路由。这些信息分别位于流file，内容和出处存储库中。

在RouteonAttribute处理器检查FlowFile的状态代码属性之后，它确定应将其路由到两个不同的位置。发生的第一件事是处理器克隆流flefile以创建F2。这复制了所有属性和指向内容的指针。由于它只是路由和分析属性，因此内容不会改变。然后将流件添加到相应的连接队列中，以等待下一个处理器检索它们以进行处理。

Provenancereporter记录了发生的更改，其中包括克隆和两个路线事件。这些事件中的每一个都有指向相关内容的指针，并以快照的形式包含相应的流文件的副本。




## 扩展路由用例

除了基于属性的路由流文件外，某些处理器还基于内容路由。虽然它的效率不高，但有时是必要的，因为您想将流文件的内容分为多个流files。

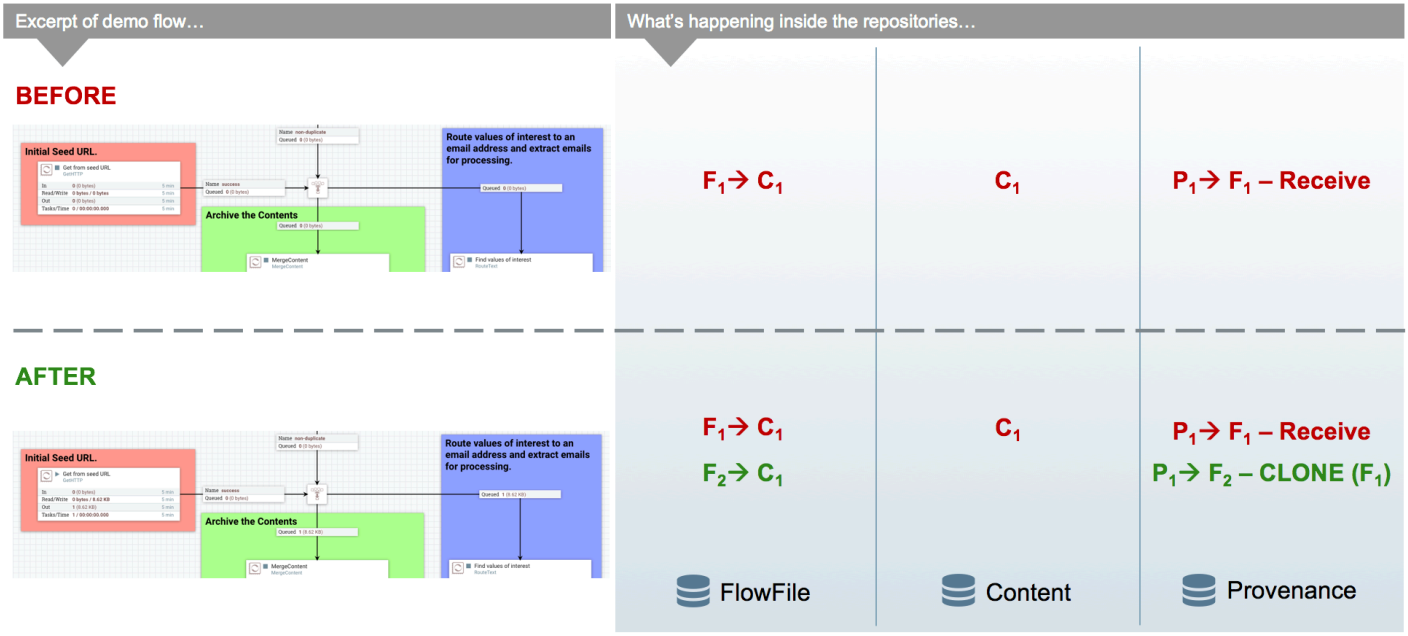
一个示例是Splittext处理器。该处理器分析了寻找终点字符的内容，并创建了包含可配置数量线的新FlowFiles。 Web爬网流利用它将潜在的URL分为单行以进行URL提取，并充当InvokeHTTP的请求。 Splittext处理器的一个好处是，由于处理器将连续的块（没有流file内容是不相交或重叠的），因此处理器可以执行此路由而无需复制任何内容。它所做的只是创建新的FlowFiles，每个FlowFiles都有指向原始FlowFile内容的一部分的指针。 NIFI API中内置的内容分界和拆分设施使它成为可能。虽然在可行的情况下，并不总是以这种方式分裂，但性能优势是相当大的。

RuteText是一个处理器，它显示了为什么需要复制某些路由样式的内容。该处理器分析每行，并根据可配置的属性将其路由到一个或多个关系。当将多个线路路由到相同的关系（对于相同的输入流文件）时，这些行被合并到一个流file中。由于行可能是不相交的（第1行和100行的相同关系路线），并且一个指针无法准确描述流文件的内容，因此处理器必须将内容复制到新位置。例如，在Web crawler流中，RuteText处理器将包含“ NIFI”的所有线路路由到“ NIFI”关系。因此，如果在网页上多次具有“ NIFI”的输入FlowFile时，将仅发送一封电子邮件（通过后续的Putemail处理器）。

## funnels

该漏斗是从一个或多个连接中获取输入并将其路由到一个或多个目的地的组件。用户指南中描述了其典型用例。无论用用例如何，如果漏斗下游只有一个处理器，则漏斗没有发出的出处事件，并且在出处图中似乎是看不见的。如果有多个下游处理器，例如WebCrawler中的一个处理器，则会发生克隆事件。参考下图时，您可以看到新的流file（F2）是从原始流file（F1）克隆的，就像上面的路由一样，新的FlowFile只是指向相同内容的指针（内容未复制）。

从开发人员的角度来看，您可以将漏斗视为非常简单的处理器。当计划运行时，它只需进行“ processSession.get（）”，然后对输出连接进行“ processSession.transfer（）”。如果有一个以上的输出连接（如下面的示例），则运行“ processSession.clone（）”。最后，称为“ processSession.commit（）”，完成了交易。

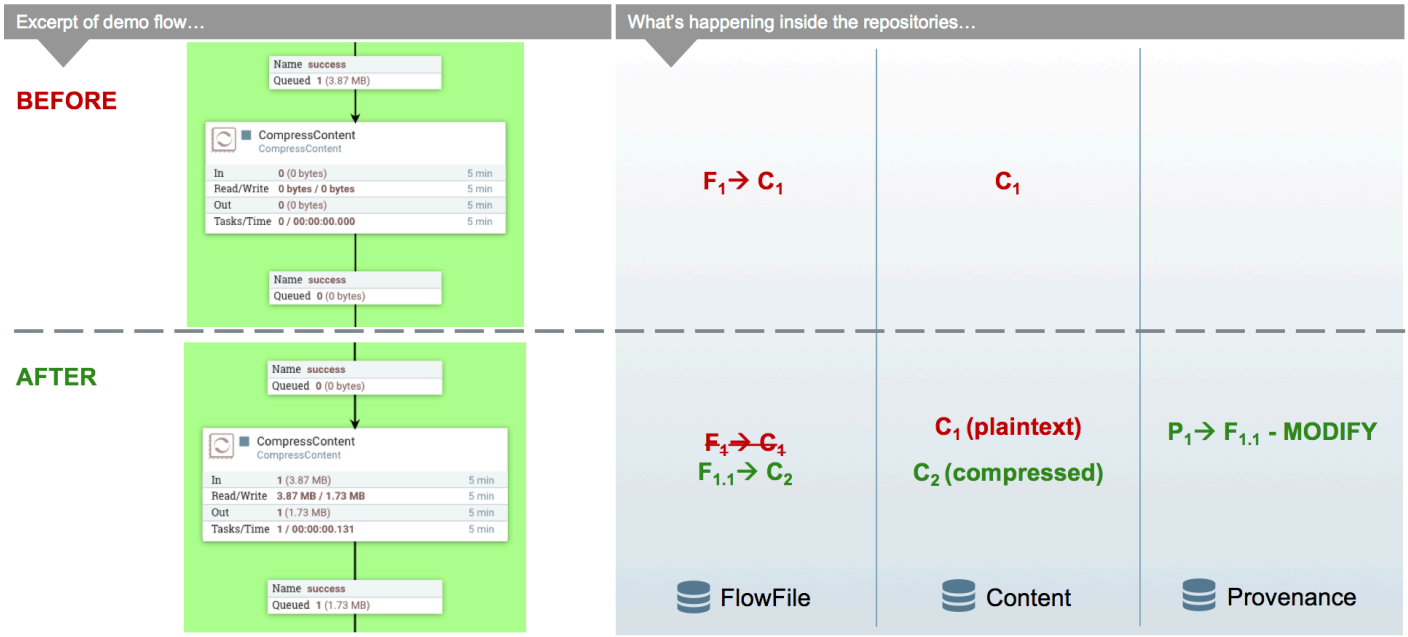



## 复制写

在上一个示例中，只有路由，但没有对流file的内容进行更改。下一个示例着重于压缩处理器，该处理器压缩了包含排队的网页的合并流文件捆绑，这些流量已排队进行分析。

在此示例中，FlowFile F1的内容C1在压缩符号处理器中被压缩。由于C1是不可变的，我们想要完整的可重复出处历史记录，因此我们不能仅仅覆盖C1。为了“修改” C1，我们执行“写入副本”，我们通过修改内容将其复制到内容存储库中的新位置来完成。这样做时，FlowFile参考F1将更新以指向新的压缩内容C2，并创建了新的出处事件P2引用新的FlowFile F1.1。由于FlowFile Repo是不可变的，而不是修改旧的F1，而是创建了一个新的Delta（F1.1）。以前的出处事件仍然具有指向内容C1并包含旧属性的指针，但它们不是流filefile的最新版本。

为了专注于写入事件的副本，省略了flowfile的（F1）出处事件。




## 写入用例上的扩展副本

在写作上复制的独特情况是MemgeContent处理器。几乎每个处理器一次仅在一个流纸上起作用。MergeContent处理器的独特之处在于它吸收了多个流量文件并将它们组合为一个。当前，MergeContent具有多个不同的合并策略，但所有策略都要求将输入流件的内容复制到新的合并位置。在获得合并后，它发出了“ JOIN”类型的出处事件，该事件确定给定父母共同创建了一个新的孩子FlowFile。

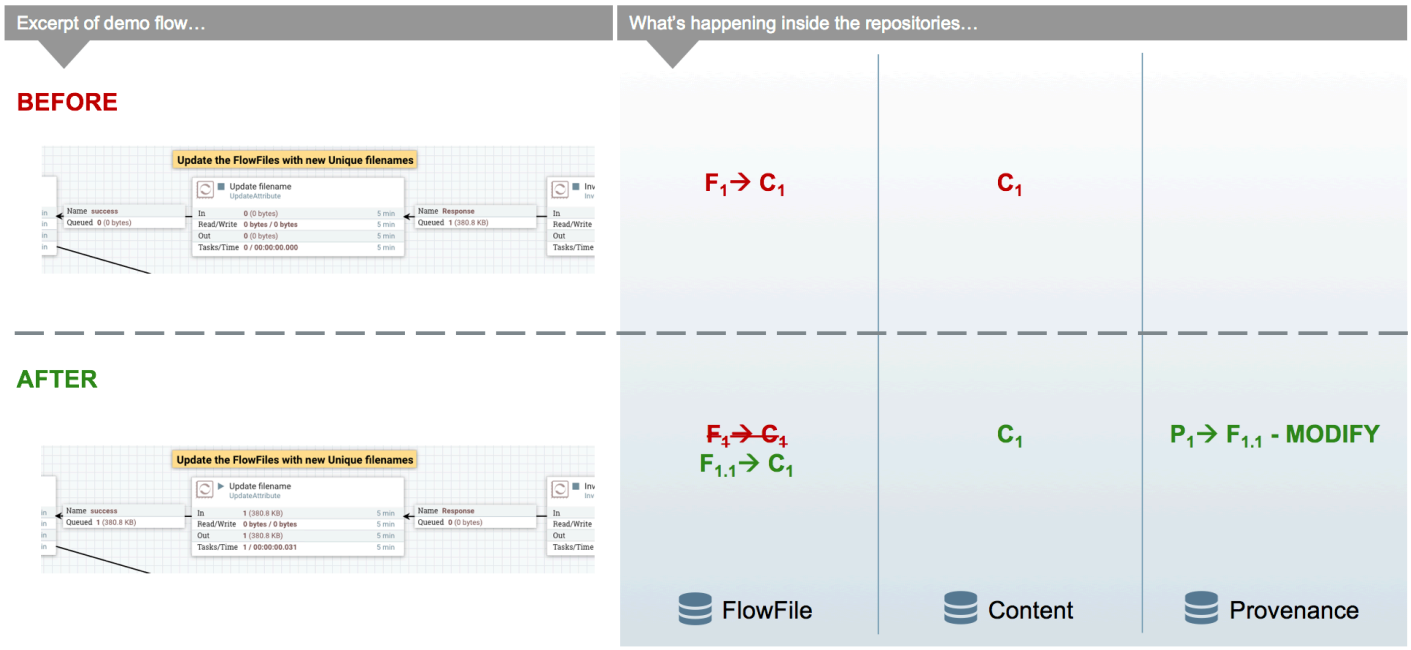
## 更新属性

使用FlowFile的属性是NIFI的核心方面。假定属性足够小，每当处理器上执行时，都可以将其完全读取到本地内存中。因此，重要的是要易于使用。由于属性是路由和处理流文件的核心方式，因此只有更改流file属性的处理器非常常见。一个这样的示例是UpdateTatribute处理器。所有的UpdateAttribute处理器所做的就是根据处理器的属性更改传入FlowFile的属性。

看一下图表，在处理器之前，有具有属性和指向内容的指针（C1）的FlowFile（F1）。处理器通过创建一个仍有指向内容的指针（C1）指针的新的增量（F1.1）来更新FlowFile的属性。当发生这种情况时，会发出“属性_MODIFIED”出处事件。

在此示例中，以前的处理器（InvokeHTTP）从URL获取了信息，并创建了带有与请求FlowFile相同的文件名属性的新响应FlowFile。这无助于描述响应流量文件，因此UpdateAttribute处理器将文件名属性修改为更相关的内容（URL和Transaction ID）。

为了专注于属性_修改事件，省略了flowfile的（F1）出处事件，直到这一点。




## 典型的用例注释

除了通过UpdateAttribute添加任意属性外，从流file的内容中提取信息为属性是一个非常常见的用例。 Web爬网流中的一个这样的示例是ExtractText处理器。当URL嵌入流file的内容中时，我们无法使用它，因此我们将URL从流file的内容物中提取并将其作为属性提取。这样，我们可以使用表达式语言在InvokeHTTP的URL属性中引用此属性。

# 数据出口

最终，NIFI中的数据将到达将其加载到另一个系统中，我们可以停止处理它，或者我们过滤了流file并确定我们不再关心它。无论哪种方式，流文件最终都会被“删除”。 “ Drop”是出处事件，这意味着我们不再处理流程中的流file，并且可用于删除。它保留在流文件存储库中，直到下一个存储库检查点为止。出处存储库将“ nifi.properties”（默认值为24小时）中的大量时间保持出处事件。一旦FlowFile离开NIFI，就会将内容存储库中的内容标记为删除，并且会出现写入对数的背景检查点处理以紧凑/删除。除非另一个FlowFile引用相同的内容，或者在“ Nifi.properties”中启用了归档。如果启用了归档，则存在内容，直到达到最大磁盘百分比或达到最大保留期（也设置在'nifi.properties'中）。

## 更深的视图：检查点后删除

本节严重依赖上面“更深视图：内容声明”部分的信息。

一旦将“ .partial”文件与基础存储机构同步，并将其重命名为新快照（在“流file repo”部分中详细介绍），将对FlowFile Repo进行回调，以释放所有旧内容声明（这是在检查点后完成的，以便在检查点后不会丢失内容，如果出错了出错）。 FlowFile Repo知道可以发布哪些内容声明并通知资源索赔经理。资源索赔经理会跟踪已发布的所有内容索赔以及准备删除哪些资源索赔（当不再有任何流载文件引用该流程中的流量文件时，就可以删除资源索赔）。

内容回购定期询问资源索赔经理，可以清理哪些资源索赔。然后，内容回购做出决定是否应归档或删除资源索赔（基于'nifi.properties'文件中的“ nifi.content.repository.archive.enabled”属性的“ nifi.content.repository.archive.enabled”的值）。如果禁用归档，则简单地将文件从磁盘中删除。否则，背景线程运行以查看何时应删除档案（基于上述条件）。该背景线程将保留10,000个最古老的内容索赔的列表，并将其删除直至低于必要的阈值。如果内容用完了，则声称它会扫描回购中最古老的内容以重新填充列表。这提供了一个模型，该模型在Java Heap利用率以及磁盘I/O使用方面都具有有效的效率。

### 关联不同的数据

出处存储库的功能之一是，它允许有效访问依次发生的事件。然后，NIFI报告任务可以用来迭代这些事件并将其发送到外部服务。如果其他系统也向该外部系统发送类似类型的事件，则可能有必要将NIFI FlowFile与另一个信息相关联。例如，如果使用GetSFTP来检索数据，则使用其独特的UUID引用该流file。但是，如果将文件放置在文件名中的文件中，则NIFI应该具有一种机制来表明这些是同一数据。这是通过调用provenancere.associate（）方法来实现的，并提供流file的uuid和备用名称（在此示例中为文件名）。由于确定两个数据相同的确定可能与流有关，因此数据流管理器通常有必要建立此关联。这样做的一种简单方法是使用UpdateAttribute处理器，并将其配置为设置“备用.Identifier”属性。这将自动排放“助理”事件，使用任何值添加为“备用.Identifier”属性。

## 闭幕词

NIFI利用副本，通过参考和不变性概念，是一个快速，高效且强大的企业数据流平台。该文档涵盖了可插入界面的特定实现。其中包括基于写入日志的书面文件存储库，基于文件的出处存储库和基于文件的内容存储库的实现。这些实现是NIFI默认值，但可以插入，因此，如果需要，用户可以自己编写以实现某些用例。

希望该文档能使您更好地了解NIFI的低级功能及其背后的决策。如果您想对某些内容进行更深入的解释，否则您认为应该包括一些内容，请随时向Apache Nifi开发人员邮件列表发送电子邮件（dev@nifi.apache.org）。

最后更新2025-01-24 16:40:11 +0100