集群 Mendix Runtime

1 简介

本页介绍将 Mendix Runtime 作为集群运行的行为和影响。使用集群功能，可以将 Mendix 应用程序设置为在负载均衡器后运行，以启用故障转移和/或高可用性架构。

支持集群的主要功能是 Mendix 的无状态运行时架构。这表示脏状态（不可保持的实体实例和尚未保持的更改）存储在客户端而不是服务器上。这样可以更轻松地扩展 Mendix Runtime，因为每个集群节点都可以处理来自客户端的任何请求。通过无状态运行时架构，还可以更好地维护脏状态，从而更好地洞察应用程序的状态。

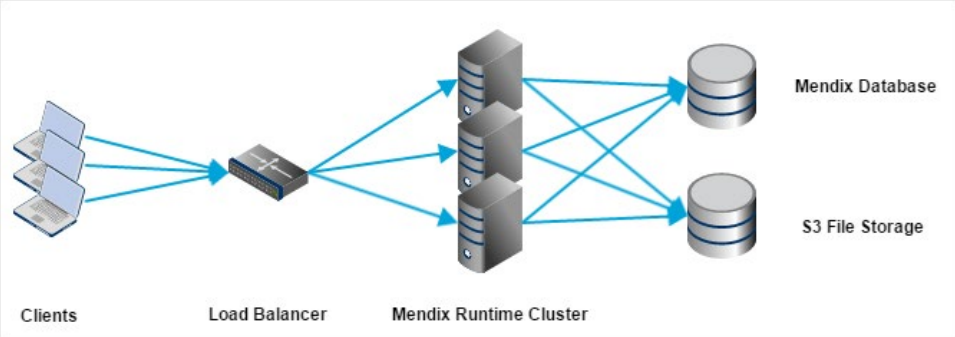
2 集群支持

我们的 Cloud Foundry 构建包实施本身内置了集群支持。这就意味着仅用 Cloud Foundry 即可进行扩展。此构建包可确保系统自动作为集群启动。

Kubernetes 也支持集群，前提是必须使用 *StatefulSet*。

3 集群架构

Mendix Runtime 集群需要以下架构：



这表示 Mendix 集群需要负载均衡器来将客户端的负载分布到可用 Runtime 集群节点上。也即所有节点都需要连接到同一个 Mendix 数据库，并且所有文件都需要存储在 S3 上（有关详细信息，请参见以下“文件存储”部分）。集群中的节点数取决于应用程序、高可用性要求及其使用情况。

4 集群领导与集群从属

Mendix Runtime 具有集群领导的概念。它是执行集群管理活动的 Mendix Runtime 集群内的单个节点。这些活动包括：

- **会话清理处理** - 各节点将会话设置为过期（即不用于已配置的时间跨度），并且集群领导会移除保存在数据库中的会话
    - 在特殊情况下（例如节点崩溃），有些会话可能不会从数据库中移除，在这种情况下，集群领导仍然可以确保移除这些会话
- **集群节点过期处理** - 在集群节点过期后移除相应集群节点（即不会为配置的时间跨度提供检测信号）
- **后台作业过期处理** - 在信息过期后（即比特定时间跨度要早）移除有关后台作业的数据
- **解锁被封锁的用户**
- **执行计划事件** - 计划事件仅在集群领导上执行
- **进行新部署后执行数据库同步**
- **进行新部署后清除保持的会话** - 使所有现有会话无效，从而与最新的模型版本同步

这些活动仅由集群领导执行。如果集群领导未运行，集群仍会工作，但不会执行上述活动。

由 Cloud Foundry 构建包决定哪些节点作为集群领导和集群从属。

5 集群启动

可以启动和停止集群中的单个节点，而不影响应用程序的正常运行时间。但是，在部署新版应用程序时，将重启整个集群，并由集群领导决定是否需要同步数据库。这表示在此操作完成后部署应用程序时会停机一段时间。

如果需要同步数据库，所有集群从属都将等待集群领导完成数据库同步。数据库同步完成后，所有集群节点都将完全正常运行。

如果不需要同步数据库，所有集群节点在启动后即可完全正常运行。

6 文件存储

上传的文件应存储在共享文件存储设备中，因为每个 Mendix Runtime 节点都应访问相同的文件。可以共享本地存储设备，也可以将文件存储在诸如 Amazon S3 文件存储、Microsoft Azure Blob 存储或 IBM Bluemix Object Storage 等中央存储设备中。

更多有关配置 Mendix Runtime 以在这些存储设备上存储文件的信息，请参见“运行时自定义”部分。

7 启动后与关闭前微流

在 Mendix 中，可能要配置启动后和关闭前微流。在 Mendix 集群中，这表示这些微流是按节点调用的。这样您就可以注册请求处理程序和其他活动。但是，强烈建议不要在这些微流期间进行数据库维护，因为这可能会影响同一集群的其他节点。集群启动或关闭时无法运行微流。

8 集群限制

8.1 微流调试

在运行多节点集群时，无法预测微流将在哪个节点上执行。因此，无法通过 Mendix Studio Pro 在集群中调试此类微流执行。但是，在运行 Mendix Runtime 的单个实例时，仍可以调试微流。

8.2 集群范围内锁定（保证单次执行）

某些应用程序要求保证在特定时间点单次执行一定活动。在单个节点 Mendix Runtime 中，这可以通过使用 JVM 锁得到保证。但是，在分布式场景中，这些 JVM 在不同的机器上运行，因此没有可用的锁定系统。Mendix 也不支持集群范围内锁定。如果无法避免这种情况，可能需要采用外部分布式锁管理器。但是，请记住，在分布式系统中，锁定十分复杂，而且容易发生故障（例如，出现锁定导致的饥饿或过期）。

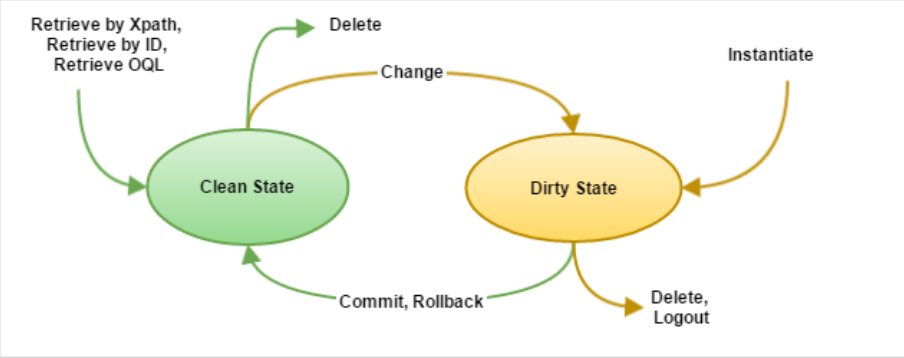
由于上述原因，微流的**不允许同时执行**属性仅适用于单个节点。

9 集群中的脏状态

当用户登录 Mendix 应用程序并开始体验某一应用程序流时，系统可以临时保留某些数据，同时不将数据保存在数据库中。这些数据保留在 Mendix Client 内存中，并代表用户传达至 Mendix Runtime 节点。

例如，假设您正在通过 Mendix 应用程序为假期预订航班、酒店和出租车。前三步依次是选择和安排航班、酒店和出租车，最后一步则是确认预订和付款。每一步可能是在不同的屏幕中进行，但从第一步到第二步时，您仍希望记住自己预订的航班。我们称之为“脏状态”。数据尚未最终确定，但应在不同请求之间保留。这是因为必须可靠地向外扩展和支持故障转移场景，无法在请求之间将状态存储在单个 Mendix Runtime 节点的内存中。因此，状态将返回到调用者 (Mendix Client) 并添加到后续请求中，以便每个节点都可以处理这些请求的相应状态。

下图描述了此行为：



读取对象和来自 Mendix 数据库的删除（未更改的）对象仍处于“干净状态”。更改现有对象或实例化新对象都将创建“脏状态”。脏状态需要随每次请求从 Mendix Client 发送至 Mendix Runtime。提交对象或回滚都会将其从脏状态移除。如果删除实例化或已更改的对象，也会发生相同的情况。不可保持的实体始终是脏状态一部分。

在各请求之间，只能保留源自 Mendix Client（同步和异步调用）的请求的脏状态。对于所有其他请求（例如计划事件、Web 服务或后台执行），仅留存当前请求的状态。之后，必须保持或丢弃脏状态。仅允许 Mendix Client 请求保留脏状态的原因是，这是当前处理实际用户输入的唯一通道。用户输入要求请求之间的数据具有更高的互动性和灵活性。通过

仅允许这些请求保留脏状态，不仅可以尽量减少 Mendix Runtime 和外部源上的负载，而且可以优化性能。

每次重启 Mendix Client 时，所有状态都将被丢弃，因为其仅保留在 Mendix Client 内存中。如果重新加载浏览器选项卡（例如按 F5 时）、重新启动移动混合应用程序或明确退出，将重启 Mendix Client。

作为脏状态一部分的对象越多，必须在 Mendix Runtime 和 Mendix Client 之间的请求和响应中传递的数据也就越多。这些都会影响性能。在集群环境中，建议最小化脏状态的体量，从而尽量减少同步对性能的影响。

Mendix Client 尝试通过仅发送处理请求时可能读取的数据来优化发送至 Mendix Runtime 的状态量。例如，如果调用的微流将得到的 `Booking` 作为参数并检索相关 `Flight`，则客户端将仅传递来自随请求产生的脏状态的 `Booking` 和相关 `Flight`，而不是 `Hotel`。请注意，此行为是最佳做法；如果微流太复杂而无法进行分析（例如，在调用 Java 操作时，将状态对象作为参数），整个脏状态将一起发送。可以通过优化网络调用项目设置禁用此优化。

必须意识到，在调用 Mendix 中外部网络服务以获取外部数据时，这些操作的响应将转换为 Mendix 实体。这些实体只要未保持在 Mendix 数据库中，就会成为脏状态的一部分，并且会对应用程序的性能产生负面影响。为了减少这种影响，此行为将来可能会发生变化。

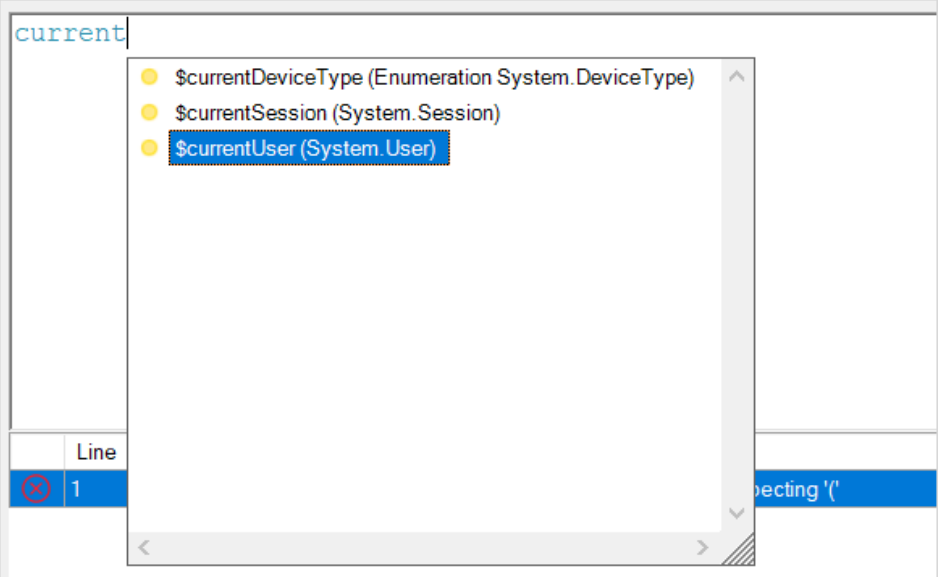
为了减少大型请求和响应对性能的影响，应用程序开发人员应了解以下导致大型请求和响应的场景：

- 创建大量不可保持的实体并将其显示在页面中的微流
- 调用网络服务以检索外部数据并将其转换为不可保持的实体的微流
- 具有多个微流数据源数据视图的页面，每个视图都会将状态传递至 Mendix Runtime 以处理微流

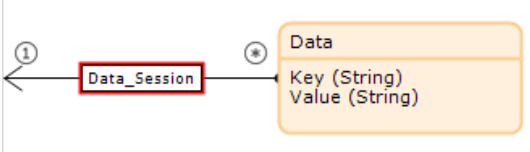
为了确保在应用程序中出现上述场景时，脏状态不会变得过大，建议明确删除非必要对象，使其不再是状态的一部分。这将释放 Mendix Runtime 节点的内存，从而处理请求并提高性能。

10 将实体与 System.Session 或 System.User 关联

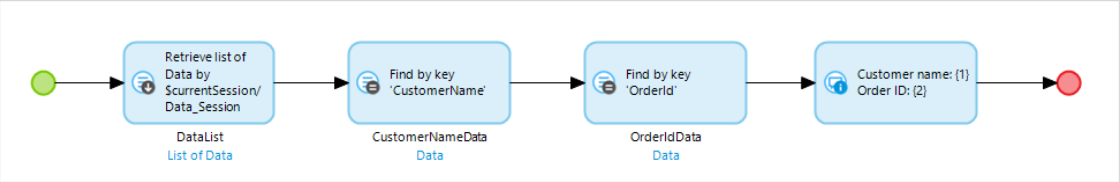
微流中提供了 $currentSession *Session* 对象，从而可以轻松获得对当前会话的引用。如果需要存储对象，其关联可以设为 $currentSession；如果需要再次检索对象，$currentSession 可以用作通过关联检索所需对象的起点。可以对关联对象进行设计，使其满足预期需求。此模式同样适用于与 System.User 相关联的实体。在该情况下，可以使用 $currentUser *User* 对象。



例如，可以将 Key 和 Value 成员添加到与 System.Session 相关联的 Data 实体中（从而获得键值常数）。



通过对 Data 实例列表的 Key 值执行查找操作，可轻松获得 Value 值。



如果数据与当前用户或当前会话相关联，则无法自动对其进行垃圾回收。因此，数据将与每次请求一起发送至服务器，并通过相应请求的响应返回。因此，如果没有其他解决方案可用于保留临时数据，则应将实体实例与当前用户和当前会话相关联。

11 会话始终保持

为了支持无缝集群，会话始终保持在数据库中。在以前的版本中，这是一个已知的性能瓶颈。为了减轻此性能影响，Mendix 现在包含了相应优化。

为此，通过为保持的会话指定 30 秒（默认情况下）的最大缓存时间，可以减少往返数据库的次数。这表示，退出会话后，仍可在集群的其他节点上访问会话 30 秒，但前提是该节点在注销前处理了与该会话有关的之前请求。此超时可以配置。减少上述超时可使集群更加安全，因为仍可以在配置的时间窗中访问会话的可能性更小。但是，这也需要更频繁地往返数据库（从而会影响性能）。增加超时则具有相反效果。这可以通过设置 `SessionValidationTimeout`（以毫秒计值）来配置。

保持的会话还会应每次请求存储最后活动日期。为了改进此性能的特定方面，会话的最后活动日期属性不再在每次请求后立即提交至数据库。相反，此信息会排入操作队列，以按可配置的间隔运行，从而存储在 Mendix 数据库中。此操作将验证会话是否尚未通过另一个节点注销，以及最后活动日期是否比数据库中的活动日期更近。可以通过设置 `ClusterManagerActionInterval`（以毫秒计值）来配置间隔。

覆盖 `SessionTimeout` 和 `ClusterManagerActionInterval` 的默认值可能会影响“保持活动”的行为，并导致意外注销会话。最佳做法是将 `ClusterManagerActionInterval` 设为 `SessionTimeout` 的一半，以便每个节点能在会话超时间隔内至少运行一次清理操作。