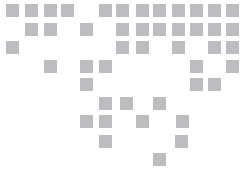

第 1 章 Chapter 1

Kafka 简介

Kafka 是一个高度可扩展的消息系统，它在 LinkedIn 的中央数据管道中扮演着十分重要的角色，因其可水平扩展和高吞吐率而被广泛使用，现在已被多家不同类型的公司作为多种类型的数据管道和消息系统。本章将聚集于 Kafka 诞生的背景、Kafka 在 LinkedIn 内部的应用、Kafka 的主要设计目标和为什么使用消息系统这四个方面，简单介绍典型的消息系统 Kafka，尤其在应用的时候需要多思考后两个方面：Kafka 的设计目标和应用层开发为什么需要使用消息系统。并以此为切入点，为之后的章节作铺垫。

1.1 Kafka 诞生的背景

对于一个高效的组织，所有数据需要对该组织的所有服务和系统是可用的，以便挖掘出数据的最大价值。数据采集和数据使用是一个金字塔的结构，底部为以某种统一的方式捕获数据，这些数据需要以统一的方式建模，以方便读取和处理。捕获数据的工作做扎实后，在这个基础上以不同方法处理这些数据就变得得心应手。

数据捕获的来源主要有两种：一种是记录正在发生的事件数据。比如 Web 系统中的用户活动日志（用户的点击选择等）、交警行业中的违章事件等。随着传统行业业务活动的数字化，事件数据正在不断增长，而且这个趋势没有停止。这种类型的事件数据记录了已经发生的事情，往往比传统数据库应用要大好几个数量级。因此对于数据的捕获、数据的处理提出了重大的挑战；另一种是经过二次分析处理之后的数据。对捕获的数据进行二次分析处理后得到的数据也需要记录保存，这里的处理指的是利用批处理、图分析等专有的数据处理系统进行了处理，这些加工后的数据可以作为数据捕获的第二个来源。

总之，捕获的数据越来越多，如何将这些巨量的数据以可靠的、完整的数据流方式传递给数据分析处理系统也变得越来越困难。

LinkedIn 使用的数据系统包括：

- 全文搜索
- Social Graph（社会图谱）
- Voldemort（键值存储）
- Espresso（文档存储）
- 推荐引擎
- OLAP（查询引擎）
- Hadoop
- Teradata（数据仓库）
- Ingraphs（监控图表和指标服务）

上述专用的分布式系统都需要经过数据源来获取数据，同时有些系统还会产生数据，作为其他系统的数据源。LinkedIn 曾尝试为每个数据源和目标构建自定义的数据加载，很显然这是不可行的。LinkedIn 有几十个数据系统和数据仓库。把这些系统和仓库联系起来，就会导致任意两两系统间构建自定义的管道，如图 1-1 所示。

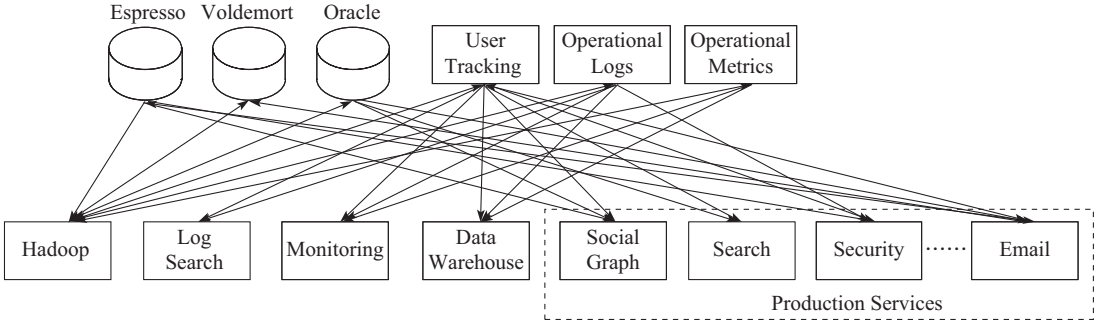

图 1-1　各系统之间的自定义管道

需要注意的是，数据是双向流动的，例如许多系统（数据库、Hadoop）同时是数据传输的来源和目的端。这就意味着我们最后要为那些系统建立两个通道：一个用于数据输入，一个用于数据输出。要避免上面的问题，我们需要如图 1-2 所示的通用方式。

我们需要尽量将每个生产者、消费者与数据源隔离。理想情况下，生产者或消费者应该只与一个数据源单独集成，这样就能访问到所有数据。根据这个思路想到增加一个新的数据系统：

- 作为数据来源或者数据目的地。
- 集成工作只需要连接这个新系统到一个单独的管道，而无须连接到每个数据的生产者和消费者。

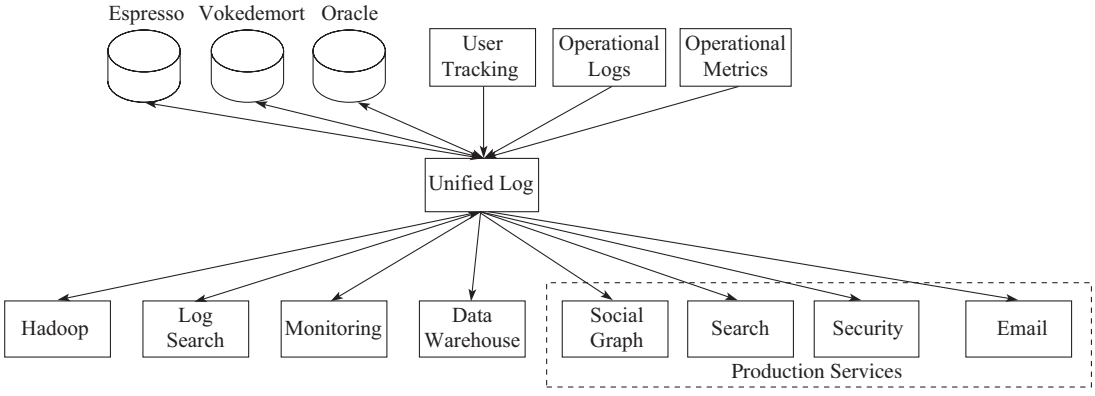

图 1-2　各系统之间的共享管道

这个新的数据系统就是 Kafka。Kafka 作为 LinkedIn 中的“中枢神经系统”，管理从各个应用程序汇聚到此的信息流，这些数据经过处理后再被分发到各处。Kafka 作为一个消息系统，进行消息的传递；同时它也是日志存储系统，以日志的形式存储了数据源的所有数据。

1.2　Kafka 在 LinkedIn 内部的应用

LinkedIn 的工程师团队已经把 Kafka 打造为管理信息流的开源解决方案。他们把 Kafka 作为消息中枢，帮助公司的各个应用以松耦合的方式在一起工作。LinkedIn 已经严重依赖于 Kafka，并且基于 Kafka 的生态系统，LinkedIn 开发出了一些开源组件和公司内部组件。

Kafka 在 LinkedIn 中的使用场景如下所述：

- **系统监控**：LinkedIn 内所有的主机都会往 Kafka 发送系统健康信息和运行信息，负责展示运维信息和报警的系统则从 Kafka 订阅获取这些运维信息，处理后进行业务的展示和告警业务的触发。更进一步，LinkedIn 通过自家的实时流处理系统 Samza 对这些运维数据进行实时的处理分析，生成实时的调用图分析。
- **传统的消息队列**：LinkedIn 内大量的应用系统把 Kafka 作为一个分布式消息队列进行使用。这些应用囊括了搜索、内容相关性反馈等应用，这些应用将处理后的数据通过 Kafka 传递到 Voldemort 分布式存储系统。
- **分析**：LinkedIn 会搜集所有的数据以更好地了解用户是如何使用 LinkedIn 的产品的。哪些网页被浏览，哪些内容被点击这样的信息都会发送到每个数据中心的 Kafka。这些数据被汇总起来并通过 Kafka 发送到 Hadoop 集群进行分析和每日报表生成。
- **作为其他分布式日志系统的组件**：Kafka 也被 LinkedIn 内其他分布式系统作为核心的日志组件，比如大数据仓储解决方案 Pinot。Kafka 也被分布式数据库 Espresso 用于负责数据的复制与修改。

除了 Kafka 的直接应用，LinkedIn 还开发了一些 Kafka 组件以应对其他的一些使用场景：

- MirrorMaker：让 Kafka 集群之间能同步数据。在很多情况下 LinkedIn 需要做跨数据中心的操作，对这些操作的事件记录，原有的 Kafka 无法支持，通过 MirrorMaker，Kafka 也能支持跨数据中心的事件记录传递。
- RESTful 接口：用户能通过 RESTful 接口向 Kafka 发布消费消息，而不需要开发 Java 代码的客户端。
- 审计服务：事件一般是在一个数据中心产生的，有时候会有场景需要在另一个数据中心对该事件进行离线分析。为此，LindedIn 会把事件消息从一个数据中心同步到另一个数据中心。在消息同步的过程中，消费消息的应用需要知道什么时候这些消息被同步完，之后应用才可以开始离线处理。审计服务保证了这一点。

1.3 Kafka 的主要设计目标

Kafka 作为一种分布式的、基于发布 / 订阅的消息系统，其主要设计目标如下：

- 以时间复杂度为 $O(1)$ 的方式提供消息持久化能力，即使对 TB 级以上的数据也能保证常数时间的访问性能。
- 高吞吐率，即使在非常廉价的商用机器上也能做到单机支持每秒 100K 条消息的传输。
- 支持 Kafka Server 间的消息分区，及分布式消费，同时保证每个分区内的消息顺序传输。
- 支持离线数据处理和实时数据处理。
- 支持在线水平扩展。

1.4 为什么使用消息系统

回过头来看一下，我们为什么需要使用消息系统呢？其大致目的如下：

- 解耦：在项目启动之初来预测将来项目会碰到什么需求，是极其困难的。消息系统在处理过程中间插入了一个隐含的、基于数据的接口层，两边的处理过程都要实现这一接口。这使得开发人员可以独立地扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束即可。
- 冗余：有些情况下，处理数据的过程会失败。除非数据被持久化，否则将造成丢失。消息队列把数据进行持久化直到数据完全处理完，通过这一方式规避了数据丢失的风险。许多消息队列所采用的“插入 – 获取 – 删除”范式中，在把一个消息从队列中删除之前，需要处理系统明确指出该消息已经被处理完毕，从而确保数据被安全保存直到使用完毕。

- **扩展性**：因为消息队列解耦了处理过程，所以增大消息入库和处理的频率是很容易的，只要另外增加处理过程即可。不需要改变代码，不需要调节参数，扩展就像调大电力按钮一样简单。
- **灵活性和峰值处理能力**：在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见，并且以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，不会因为突发的超负荷的请求而完全崩溃。
- **可恢复性**：系统的一部分组件失效时，不会影响整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。
- **顺序保证**：在大多使用场景下，数据处理的顺序都很重要。大部分消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理。Kafka 可保证一个分区内的消息是有序的。
- **缓冲**：在任何重要的系统中，都会需要不同的处理时间的元素。例如，加载一张图片比应用过滤器花费更少的时间。消息队列通过一个缓冲层来帮助任务最高效率地执行，写入队列的处理会尽量的快速。该缓冲有助于控制和优化数据流经过系统的速度。
- **异步通信**：很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理。想向队列中放入多少消息就放多少，然后在需要的时候再去处理。

1.5　本章小结

本章先讲述了 Kafka 诞生的背景及在 LinkedIn 公司中的应用，接着讲述了 LinkedIn 设计 Kafka 的主要目标，本质上最重要的就是两点：高吞吐量和可水平扩展。最后讲述了为什么需要使用消息系统，也就是应用层使用消息系统可以解决的问题，并且这也是本书的读者需要不断思考的问题。希望通过以上四个方面的介绍使得读者能够对 Kafka 形成初步的认识，同时对自己的系统进行思考。