7.1 构建数据管道时需要考虑的问题

7.1.1 及时性

7.1.2 可靠性

7.1.3 高吞吐量和动态吞吐量

7.1.4 数据格式

7.1.5 转换

7.1.6 安全性

7.1.7 故障处理能力

7.1.8 糯合性和灵活性

数据管道最重要的作用之一是解耦数据源和数据池。它们在很多情况下可能发生耦合。

临时数据管道

元数据丢失：如果数据管道允许 schema 发生变更，应 用程序各方就可以修改自己的代码，无需担心对整个 系统造成破坏。

未端处理：我们在讨论数据转换时就已提到，数据管道难免要做一些数据处理。在不同 的系统之间 移动数据肯定会碰到不同的数据格式和不同的应用场景。

7.2 如何在Connect API和客户端API之间作出选择

如果要将 Kafka 连接到数据存储系统，可以使用 Connect

如果你要连接的数据存储系统没有相应的连接器，那么可以考虑使用客户端 API或 Connect API 开发一个应用程序。

7.3 Kafka Connect

7.3.1 运行 Connect

你打算在生产环境使用 Connect来移动大量的数据，或者打算运行多个连接器，那么最好把 Connect部署在独立于 broker 的 服务器上。在所有的机器上安装 Kafka，并在部分服务器上启动 broker，然后在其他服务 器上启动 Connect。

启动 Connect进程与启动 broker差不多， 在调用脚本时传入一个属性文件即可

```
bin/connect-distributed.sh config/connect-distributed.properties
```

Connect 进程有以下几个重要的配置参数。

bootstrap.Servers:该参数列出了将要与 Connect协同工作的 broker服务器，连接器将 会向这些 broker写入数据或者从它们那里读取数据。你不需要指定集群所有的 broker, 不过建议至少指定 3 个。
group.id:具有相同 group id的 worker属于同一个 Connect集群。集群的连接器和它们 的任务可以运行在任意一个 worker上。
key.converter和 value.converter: Connect可以处理存储在 Kafka里的不同格式的数据 。 这两个参数分别指定了消息的键和值所使用的转换器。默认使用 Kafka 提供的 JSONConverter，当然也可以配置成 ConfluentSchema Registry 提供的 AvroConverter。

7.3.2 连接器示例——文件数据源和文件数据池(参考书籍)

7.3.3 连接器示例——从 MySQL到 ElasticSearch（参考书籍）

7.3.4 深入理解Connect

现在让我们深入理解每一个组件，以及它们之间是如何进行交互的。
1.连接器和任务
连接器插件实现了 Connector API,
1.1连接器
连接器负责以下 3 件事情。
API 包含了两部分内容。
· 决定需要运行多少个任务。
. 按照任务来拆分数据复制。
· 从 worker进程获取任务配置并将其传递下去。
2. worker 进程
worker进程是连接器和任务的“容器”。它们负责处理 H盯P请求，这些请求用于定义连 接器和连接器 的配置 。它们还负责保存连接器的配置、启动连接器和连接器任务，并把配 置信息传递给任务。 如果一个 worker进程停止工作或者发生崩愤，集群里的其他 worker 进程会 感知到(Kafka 的消费者协议提供了心跳检测机制)，并将崩溃进程的连接器和任务 重新分配给其他进程。如果有新的进程加入集群，其他进程也会感知到，并将自己的连接 器和任务分配给新 的进程，确保工作负载的均衡。进程还负责提交偏移量，如果任务抛出 异常，可以基于这些偏移量进行重试。
为了更好地理解 worker进程，我们可以将其与连接器和任务进行简单的比较。连接器和任 务负责“数据的移动”， 而 worker进程负责 REST API、配置管理、

可靠性、高可用性、伸 缩性和 负载均衡 。
这种关注点分离是 Connect API 给我们带来的最大好处，而这种好处是普通客户端 API 所 不具备的。有经验的开发人员都知道，编写代码从 Kafka读取数据并将其插入数据库只需 要一到两天的时间，但是如果要处理好配置、异常、 REST API、监控、部署、伸缩、失效 等问题，可能需要几个月 。 如果你使用连接器来实现数据复制，连接器插件会为你处理掉 一大堆复杂的问题。
3. 转化器和 Connect 的数据模型
数据模型和转化器是 Connect API需要讨论的最后一部分内容。 Connect提供了一组数据 API- 它们包含了数据对象和用于描述数据的 schema。例如， JDBC 连接器从数据库读 取了一个字段，并基于这个字段的数据类型创建了一个 Connect Schel'la 对象。然后使用这 些 Schema对象创建一个包含了所有数据库字段的 Struct——我们保存了每一个字段的名 字和它们的值。惊连接器所做的事情都很相似一一-从1原系统读取事件，并为每个事件生成 schema 和值(值就是数据对象本身) 。 目标连接器正好相反，它们获取 schema 和值，并使 用 schema来解析值，然后写入到目标系统。
源连接器只负责基于 Data API生成数据对象，那么 worker进程是如何将这些数据对象保 存到 Kafka 的?这个时候，转换器就派上用场了。用户在配置 worker进程(或连接器)时 可以选择使用合适的转化器，用于将数据保存到 Kafka。目前可用的转化器有 Avro、 JSON 和 String。 JSON 转化器可以在转换结果里附带上 schema，当然也可以不使用 schema，这 个是可配 的 。 Kafka 系统因此可以支持结构化的数据和半结构化的数据。连接器通过 Data API将数据返回给 worker 进程， worker进程使用指定的转化器将数据转换成 Avro对象、 JSON 对象或者字符串，然后将它们写入 Kafka。
对于目标连接器来说，过程刚好相反——在从 Kafka读取数据时， worker进程使用指定的转换器将各种格式(Avro、 JSON 或 String)的数据转换成 Data API 格式的对象，然后将 它们传给目标连接器，目标连接器再将它们插入到目标系统。
Connect API 因此可以支持多种类型的数据，数据类型与连接器的实现是相互独立的—— 只要有可用的转换器，连接器和数据类型可以自由组合。
4. 偏移量管理
worker进程的 REST API 提供了部署和配置管理服务，除此之外， worker进程还提供了 偏移量管理服务。连接器只要知道哪些数据是已经被处理过的，就可以通过 Kafka提供的 API来维护偏移量。
源连接器返回给 worker进程的记录里包含了一个逻辑分区和一个逻辑偏移量。

它们并非 Kafka 的分区和偏移量，而是源系统的分 区和偏移量。例如，对于文件源来说，分区可以 是一个文件，偏移量可以是文件里的一个行号或者字符号:而对于 JDBC 据来说，分区可 以是一个数据表，偏移量可以是一条记录的主键。在设计一个据连接器时，要着重考虑如 何对源系统的数据进行分区以及如何跟踪偏移量，这将影响连接器的并行能力，也决定了 连接器是否能够实现至少一次传递或者仅一次传递。

源连接器返回的记录里包含了源系统的分区和偏移量， worker 进程将这些记录发送给

Kafka。如果 Kafka确认记录保存成功， worker进程就把偏移量保存下来。偏移量的存储 机制是可插拔的， 一般会使用 Kafka 主题来保存。如果连接器发生崩愤并重启，它可以从 最近的偏移量继续处理数据。

目标连接器的处理过程恰好相反，不过也很相似。它们从 Kafka上读取包含了主题、分区 和偏移量信息的记录，然后调用连接器的 put() 方毡，该方告会将记录保存到目标系统里 。 如果保存成功，连接器会通过消费者客户端将偏移量提交到 Kafka上。

框架提供的偏移量跟踪机制简化了连接器的开发工作，并在使用多个连接器时保证了 一定 程度的行为一致性。

7.4 Connect之外的选择

如果架构里包含了 Kafka，并且需要连接大 量 的惊系统和目标系统，那么建议使用 Connect API 作为摄入 工 具。如果构建的系统是以 Hadoop 或 ElasticSearch 为中心的， Kafka 只是数据的来惊之 一 ，那么使用 Flume 或 Logstash 会更合适。