就变得非常简单。

例如，目前 commitlog 目录下存在如下三个文件：00000000000000000000、00000000001073741824、00000000002147483648。一条消息的物理偏移量为 1073741829，通过二分查找，很容易定位到这条消息在第三个文件中，因为 1073741829 大于 1073741824 小于 2147483648，然后用消息的物理偏移量与 1GB 进行取模运算，就可以得出它在第三个文件中的相对偏移量，从而在指定位置尝试读取 4 字节就解析出该消息的长度，然后轻松地提取一条完整的消息。

当然值得说明的是，基于磁盘的存储机制，通常需要对存储的内容进行 CRC 校验。这主要是因为磁盘的存储不可靠，可能会使存储的内容发生变化。具体的做法通常是：在存储关键信息时，利用一定的算法对这些信息算出一个校验和，与内容一道存储到文件中；在读取该条消息时，再用同样的算法对读取到的数据计算一次校验和；如果两个校验和相同，则认为数据未损坏。

不得不说，尽管 RocketMQ 是一款优秀的消息中间件，但在消息存储协议上还是存在一些缺陷。Kafka 在 V1.0 的消息格式中引入了可变长字节存储技术，其核心思想是对于数值类型的数据，特别是值较少的字段，无须将一个 `int` 类型的字段固定为 4 字节，而是可以用 1~5 字节来编码一个 `int` 类型的字段，从而节省存储空间。

2. ConsumeQueue 文件存储设计

结合 commitlog 文件的存储设计，根据消息物理偏移量定位消息将十分高效，但消息中间件基本上都遵循订阅与发布机制，消费端是需要根据主题进行消费消息的。如果在 commitlog 文件中按照主题查询消息，性能将极其低下，几乎无法正常运转。为了解决这个问题，RocketMQ 引入了 ConsumeQueue 文件，使之能根据主题快速定位消息，其结构如图 5-17 所示。

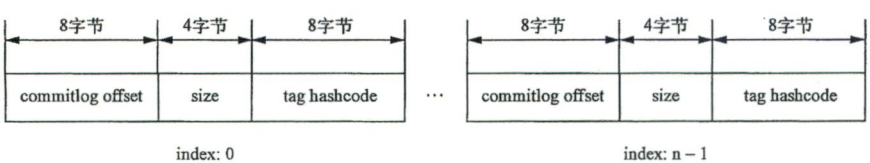

图 5-17　ConsumeQueue 文件的结构

正如 5.3.1 节中提到的，一个 ConsumeQueue 文件的长度为 300 000 × 20，其中 20 为每一个条目的长度，表示一个 ConsumeQueue 文件中存储了 300 000 个条目。让我们运用一下抽象思维：如果将一个 ConsumeQueue 文件全部加载到内存，是不是就可以将该文件看成一个数组，其中每个元素占用 20 字节，这样就可以用数组下标的方式高效地访问数据。

ConsumeQueue 中每一个条目的存储格式如下。

- 8 字节的 commitlog 偏移量，即消息的物理偏移量。