第3章

大数据采集与处理

第1章介绍了大数据技术的起源与搜索引擎对大规模网页数据存储和计算的需求密切相关，而这些网页数据几乎都来源于网络爬虫对互联网页面的持续访问、解析和下载。同时，网络爬虫在发展过程中，也逐渐使用分布式技术通过横向扩展机器资源提升工作效率，这个分布式技术是很多大数据技术的基础。

本书不会详细介绍一个每天可以稳定抓取GB级甚至TB级数据的网络爬虫应用架构。考虑到在实际的学习和工作中，网络爬虫技术应该会更多地应用于对目标数据的获取，例如，抓取微博做一个社交网络数据分析，或者抓取知乎构造一个简单的智能问答系统等。因此，在简单的介绍网络爬虫系统架构之后，把重点放在如何去抓取网页数据并进行解析，同时也会对网络爬虫所应用到的HTTP相关技术进行简要说明。

3.1 网络爬虫

3.1.1 网络爬虫介绍

网络爬虫是一种按照一定的规则，自动地抓取互联网信息的程序或者脚本。这种技术被广泛用于互联网搜索引擎或其他类似网站，可以自动采集所有爬虫能够访问到的页面内容，以获取或更新这些网站的内容和检索方式。从功能上讲，网络爬虫一般分为数据采集、处理和储存3部分。具体来说，网络爬虫会递归地对目标站点资源进行遍历：获取第一个Web页面，然后提取该页面所指向的其他页面链接。之后爬虫会访问链接指向的那些页面，继续提取那些页面的链接，以此类推。网络爬虫工作流程如图3.1所示。

(1) 首先需要获取一批种子URL集合，这个种子URL集合来自目标站点主页或者自定义添加的URL列表。

(2) 将这些种子URL放入待抓取的URL队列，该队列会在抓取过程中不断更新。对于一个多线程或者分布式的网络爬虫，这个URL队列经常通过内存数据库来维护。

(3) 网络爬虫工作时会从待抓取的URL队列中POP一个URL，解析DNS得到主机的IP地址，并将URL对应的网页下载下来，存储进已下载的网页库

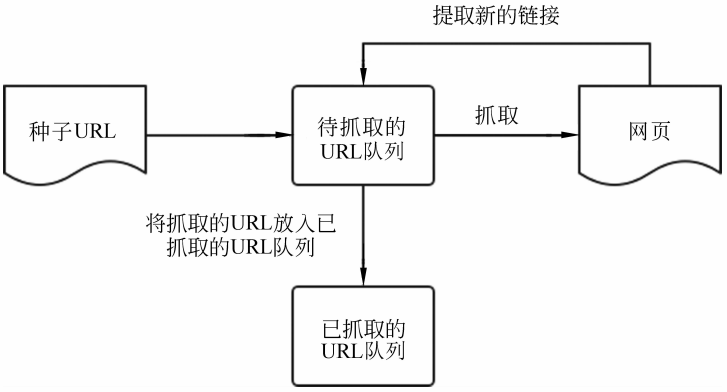

图 3.1　网络爬虫工作流程

中。之后，将抓取的 URL 放入已抓取的 URL 队列。

(4) 对于已下载的网页内容，网络爬虫会进行页面解析，提取页面中的链接，并将未抓取的 URL 放入待抓取的 URL 队列，从而进入下一个循环。

为什么网络爬虫在架构上要严格区分已抓取的 URL 队列和未抓取的 URL 队列呢？

这是因为网络爬虫在互联网上抓取页面时可能会陷入循环之中。

(1) 网络爬虫首先解析了页面 A，获取了 B 的链接。

(2) 网络爬虫获取 B 的页面并解析，得到链接 C。

(3) 网络爬虫获取 C 的页面并解析，得到链接 A。如果网络爬虫继续获取页面 A，就会陷入 A，B，C，A，B，C，…这样的循环过程当中。

因此网络爬虫在爬取页面时，必须记录已经访问过的 URL。如果要爬取全部互联网内容，记录曾经所有访问过的内容不是一件非常容易的事情，这就需要大数据存储和检索技术进行支撑。但如果只是爬取一个小型站点，那么在内存数据库(一般使用 Redis)中维护两个队列就基本能满足需求，一个队列保存待抓取的 URL 列表，一个队列保存已抓取过的 URL 列表，新提取的链接如果判断在两个队列中都不存在，就添加到待抓取的 URL 列表的末尾。

3.1.2　构建一个网络爬虫的实践经验

1. 规范化 URL

将 URL 转换成为标准形式以避免语法上的别名，例如，一些提取的链接没有站点名，或者没有 http 开头。

2. 广度优先

每次爬虫都有大量潜在的 URL 要去爬行。以广度优先的方式调度 URL 访问 Web

站点，就可以将陷入环路之中的可能性最小化。如果采用深度优先的方式，一头扎到某个站点中，就可能跳入环路，永远无法访问其他站点。

3. URL/站点黑名单

维护一个 URL/站点黑名单，然后避开黑名单中的 URL，如多次抓取失效的页面地址。

4. 内容指纹

一些更复杂的网络爬虫会使用内容指纹这种更加强大的方式检测重复。使用内容指纹的爬虫会获取页面内容中的字节，计算出一个校验和。这个校验和是页面内容的压缩表示形式。如果爬虫获取了一个页面，而此页面的校验和在已抓取的列表当中，它就不会解析页面进行新的链接发现。校验和函数往往使用像 MD5 这样的散列函数，这样两个不同页面拥有相同校验和的概率非常低。

3.1.3 HTTP 介绍

网络爬虫其实与其他 HTTP 客户端程序(浏览器)并没有本质区别，因此它们也要遵守 HTTP 规范中的规则。一些简单的反爬虫策略也是通过检查爬虫访问与浏览器访问的区别识别请求访问页面的来源是一个网络爬虫还是一个浏览器。所以，如果想让网络爬虫不会被目标站点很快地封禁，需要了解一些 HTTP 的内容和 HTTP 客户端的知识。

1. URL

URL 说明了资源位于服务器的地址。该地址通常像一个分级的文件路径，如 https://www.dlut.edu.cn/xxgk/xxjj.htm，说明了大连理工大学的学校简介位于服务器 xxgk 目录(或虚拟目录)下的 xxjj.htm 文件。

2. 参数

有时为了向应用程序提供它们所需的输入参数，以便正确地与服务器进行交互，URL 中有一个参数组件。这个组件就是 URL 中的键/值对列表，用字符“?”将其与 URL 中的其余部分分隔。它们为应用程序提供了访问资源所需要的所有附加信息。

3. URL 编码

为了避开字符集表示法带来的限制，人们设计了一种编码机制，用来在 URL 中表示各种不安全的字符，这种编码机制称为转义。转义的方法是一个百分号(%)后面跟着两个表示字符(ASCII 码的十六进制数)。因此，当在 URL 中看到这种奇怪的编码时，并不用担心。

4. HTTP 报文

HTTP 报文是在 HTTP 应用程序之间发送的数据块。这些数据块以一些文本形式

的元信息(meta-information)开头，这些信息描述了报文的内容及含义，后面跟着可选的数据部分。每条报文都包含一条来自客户端的请求，或者一条来自服务器的响应。它们由 3 部分组成：对报文进行描述的起始行(start line)，包含属性的首部(header)块，以及可选的、包含数据的主体(body)。

对网络爬虫来说，一次数据获取就是发送请求报文并下载响应报文的主体内容，该主体内容往往是 HTML 文件。因此，对 HTTP 报文的了解有助于分析网络爬虫的工作效率并进行访问策略的优化。HTTP 报文基本结构如图 3.2 所示。

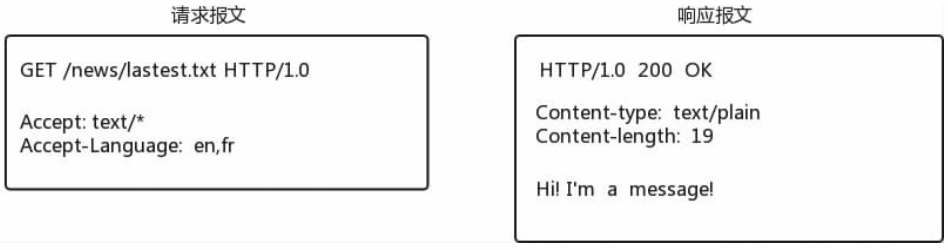

图 3.2　HTTP 报文基本结构

HTTP 首部字段(headers)向请求报文和响应报文中添加了一些附加信息。本质上，它们只是一些键/值对列表，但 HTTP 首部字段是反爬虫策略最简单的校验目标：如果没有首部字段，那么访问来源非常有可能是一个网络爬虫而不是一个浏览器。因此，在网络爬虫程序中，最好每次请求都带上对应的首部字段，其结构如表 3.1 所示。

表 3.1　HTTP 首部字段结构

首 部 实 例	描　　述
Date：Tue，15ct 2009 15:16:03 GMT	服务器产生响应的日期
Content-length：12080	实体的主体部分包含了 12 080B 的数据
Content-type：image/gif	实体的主体部分是一个 GIF 格式的图片
Accept：image/gif，image/jpeg，text/html	客户端可以接收 GIF 格式的图片和 JPEG 格式的图片以及 HTML

5. HTTP 方法

(1) GET。HTTP GET 是最常用的方法，通常用于请求服务器发送某个资源。客户端用 GET 方法向服务器发起一次 HTTP 请求，服务器响应了一段消息(message)给客户端。这个方法也是网络爬虫最常用的数据获取手段。

(2) POST。POST 方法起初是用来向服务器输入数据的。但实际上，服务器通常会用它支持 HTML 的表单。表单中填好的数据通常会被送给服务器，然后由服务器将其发送到它要去的地方。同时，很多服务器会利用 POST 方法封装请求参数，因此这个方法是 GET 之外网络爬虫最常用的方法。

6. HTTP 常见状态码

- 200 ok：客户端请求成功。
- 204 No Content：请求处理成功，但没有资源可返回。204 不允许返回任何实体的主体。
- 206 Partial Content：客户发送了一个带有 Range 头的 GET 请求，服务器完成。例如，使用 Video 播放视频，返回 206。
- 301 Moved Permanently：永久重定向。该状态码表示请求的资源已被分配了新的 URI，以后应按 Location 首部字段提示的 URI 重新保存。
- 302 Found：和 301 Moved Permanently 状态码相似，但 302 状态码代表的资源不是被永久移动的，只是临时性质的。
- 303 See Other：303 状态码和 302 Found 状态码有着相同的功能，但 303 状态码明确表示客户端应当采用 GET 方法获取资源。
- 400 Bad Request：请求报文中存在语法错误。当该错误发生时，需要修改请求的内容后再次发送请求。
- 401 Unauthorized：返回含有 401 的响应必须包含一个适用于被请求资源的 WWW-Authenticate。首部用于质询(challenge)用户信息。当浏览器初次接收到 401 响应，会弹出认证用的对话窗口。
- 403 Forbidden：该状态码表明对请求资源的访问被服务器拒绝。服务器端不需要给出拒绝的详细理由。未获得文件系统的访问授权，访问权限出现某些问题(从未授权的发送源 IP 地址试图访问)等都可能是发生 403 的原因。
- 404 Not Found：该状态码表明服务器上无法找到请求的资源。
- 500 Internal Server Error：服务器本身发生错误，也有可能是 Web 应用存在的程序漏洞或某些临时的故障。
- 503 Service Unavailable：该状态码表明服务器暂时处于超负荷或正在进行停机维护，现在无法处理请求。

3.1.4 网页解析与 CSS 选择器

网页解析其实就是从服务器返回的网页内容中提取想要的数据的过程，这个过程可以使用字符串匹配或正则表达式。但现在绝大多数网页源代码都是用 HTML 语言写的，使用字符串匹配或正则表达式的方法效率极低。所以就出现了从 HTML 代码中提取特定文本的工具包，即网页解析库。当然，这些网页解析库并不是把字符串解析和正则表达式进行封装，而是通过更好地解析语法进行页面解析。一种现在常用的方法是 CSS 选择器，包括如下子选择器。

1. 标签选择器

这种基本选择器会选择所有匹配给定元素名的元素。

语法：

```
elename
```

例如：input 将会选择所有的 <input> 元素。

2. Class(类)选择器

这种基本选择器会基于类属性的值选择元素。

语法：

```
.classname
```

例如：.index 会匹配所有包含 index 类的元素（类似于这样的定义 class＝"index"）。

3. id 选择器

这种基本选择器会选择所有 id 属性与之匹配的元素。需要注意的是，一个文档中每个 id 都应该是唯一的。

语法：

```
#idname
```

例如：#toc 将会匹配所有 id 属性为 toc 的元素（类似于这样的定义 id＝"toc"）。

4. 通用选择器

这个基本选择器选择所有节点。它也常和一个名词空间配合使用，用来选择该空间下的所有元素。

语法：

```
* ns|* *|*
```

例如：*（通配符)将会选择所有元素。

5. 属性选择器

这个基本的选择器根据元素的属性进行选择。

语法：

```
[attr] [attr=value] [attr~=value] [attr|=value] [attr^=value] [attr$=value]
[attr*=value]
```

例如：[autoplay] 将会选择所有具有 autoplay 属性的元素(不论这个属性的值是什么)。

3.1.5　项目实践 3：抓取网页并提取标题和正文

对于广告系统，判断用户的兴趣和意图往往是基于用户的浏览行为进行分析，因此需要网络爬虫对用户访问的 URL 进行页面抓取并解析页面内容，之后交给自然语言处理

模块和数据分析模块，进行进一步的数据分类或者聚类。

1. 抓取网页内容

抓取网页内容的方法很多，这里推荐使用 Python 的 requests 工具包。相比 urllib2 或 urllib3，requests 在接口封装上更加抽象和友好，用户不需要关注太多网络连接底层方面的事情，只需要调用 requests 提供的高级 API 即可完成任务。

1）安装依赖工具包

```
>pip3 install requests
```

2）requests 工具包介绍

一开始要导入 requests 模块：

```
1. import requests
```

然后，尝试获取某个网页。本例尝试获取 Github 的公共时间线。现在，有一个名为 r 的 Response 对象，我们可以从这个对象中获取所有想要的信息。

```
1. r=requests.get('https://api.github.com/events')
```

Requests 简便的 API 意味着所有 HTTP 请求类型都是显而易见的。例如，可以发送一个 HTTP POST 请求：

```
1. r=requests.post('http://httpbin.org/post', data={'key':'value'})
```

如果想为请求添加 HTTP 头部，只要简单地传递一个字典给 headers 参数即可。服务器的反爬虫系统会拒绝没有 HTTP 头部的请求，因此带上 headers 参数抓取网页数据总是一个很好的选择。

```
1. headers={'user-agent': 'my-app/0.0.1'}
2. r=requests.get(url, headers=headers)
```

通常，要发送一些编码为表单形式的数据——非常像一个 HTML 表单，只需简单地传递一个字典给 data 参数。数据字典在发出请求时会自动编码为表单形式：

```
1. payload={'key1': 'value1', 'key2': 'value2'}
2. r=requests.post("http://httpbin.org/post", data=payload)
```

可以检测响应状态码：

```
1. r=requests.get('http://httpbin.org/get')
2. r.status_code
```

如果某个响应中包含 cookies，可以快速访问它们：

```
1. url='http://example.com/some/cookie/setting/url'
2. r=requests.get(url)
3. r.cookies['example_cookie_name']
```

要想发送 cookies 到服务器,可以使用 cookies 参数:

```
1. url='http://httpbin.org/cookies'
2. cookies=dict(cookies_are='working')
3. r=requests.get(url, cookies=cookies)
```

3) 获取新浪新闻首页

```
1. import requests
2. url="https://news.sina.com.cn "
3. html=requests.get(url)
4. print(html)
```

输出结果:

```
<Response [200]>
```

2. 解析网页内容

在抓取网页内容中,可以看到网页是由 HTML 标签构成的,如果通过字符串匹配或正则表达式的方式处理文档需要开发大量代码。所幸 Python 提供了不错的工具解析 HTML 标签,BeautifulSoup 是其中易用性比较高的一种。

1) 安装依赖工具包

```
>pip3 install beautifulsoup4
```

2) BeautifulSoup 工具包介绍

使用下面一段 HTML:

```
1. html_doc="""
2. <html><head><title>The Dormouse's story</title></head>
3. <body>
4. <p class="title"><b>The Dormouse's story</b></p>
5.
6. <p class="story">Once upon a time there were three little sisters; and their
   names were
7. <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
8. <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a>and
9. <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
10. and they lived at the bottom of a well.</p>
11.
12. <p class="story">…</p>
13. """
```

引入 BeautifulSoup,并使用 soup 的 prettify 方法查看是否正确载入:

```
1. from bs4 import BeautifulSoup
2. soup=BeautifulSoup(html_doc)
```

```
3. print(soup.prettify())
```

一些简单的使用样例：

```
1. soup.title
2. #<title>The Dormouse's story</title>
3.
4. soup.title.name
5. #u'title'
6.
7. soup.title.string
8. #u'The Dormouse's story'
9.
10. soup.title.parent.name
11. #u'head'
12.
13. soup.p
14. #<p class="title"><b>The Dormouse's story</b></p>
15.
16. soup.p['class']
17. #u'title'
18.
19. soup.a
20. #<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
21.
22. soup.find_all('a')
23. #[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
24. #  <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
25. #  <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
26.
27. soup.find(id="link3")
28. #<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

3）抓取新浪新闻页面并解析 title 和提取链接

```
1. import requests
2. from bs4 import BeautifulSoup
3.
4. url="https://news.sina.com.cn"
5. html=requests.get(url)
6. soup=BeautifulSoup(html.content, 'lxml')
7. print(soup.title)
8.
9. for link in soup.select("div.ct_t_01 h1 a"):
10.     print(link.get("href"))
```

3.2　Apache Kafka

在第 2 章的图 2.2 计算广告系统架构中，可发现有数据管道模块。一般在以往的程序设计中，这个模块很少会被使用：前端页面交互的数据或者后台程序产生的数据会直接写入文件或者数据库当中，而不是写入一个数据管道。那么为什么需要这个模块呢？

因为在大数据的场景下，数据时常是大规模高并发产生的，可以想象春节前的 12306 应用或者双十一期间的淘宝网站。一瞬间产生的海量数据是非常难被快速处理并记入数据库中的。如果不想丢失数据，就需要在数据的产生方和使用方之间构建一个缓冲器，让这个缓冲器承受大数据的压力，从而让使用方可以自如地进行数据处理而不是将数据丢掉。这个数据产生方一般称为生产者，使用方一般称为消费者。

同时，一个复杂的系统拥有众多的数据生产者和消费者，需要一个消息的发布和订阅机制代替点对点的消息传输。生产者只关心往队列里写消息，消费者只关心从队列里读消息即可。这种方式在系统架构设计中起到了模块间解耦、流量削峰和异步处理的作用。

可以说，Apache Kafka 就是为上述需求设计的，但可以提供的能力远不止消息的发布和订阅。如官网所说，Apache Kafka 是一个分布式的流式处理平台，具有高性能、持久化、多副本备份、横向扩展能力。作为一个流式处理平台，Apache Kafka 具备以下 3 个特点。

（1）发布和订阅消息。

（2）具备消息的存储和容错能力。

（3）即时处理消息。

3.2.1　系统架构

Apache Kafka 通过将生产者（producer）、代理（broker）和消费者（consumer）分布在不同的节点（机器）上，构成分布式系统架构如图 3.3 所示。主题（topic）是 Kafka 提供的高层抽象，一个主题就是一个类别或者一个可订阅的数据名称。生产者可以向一个主题

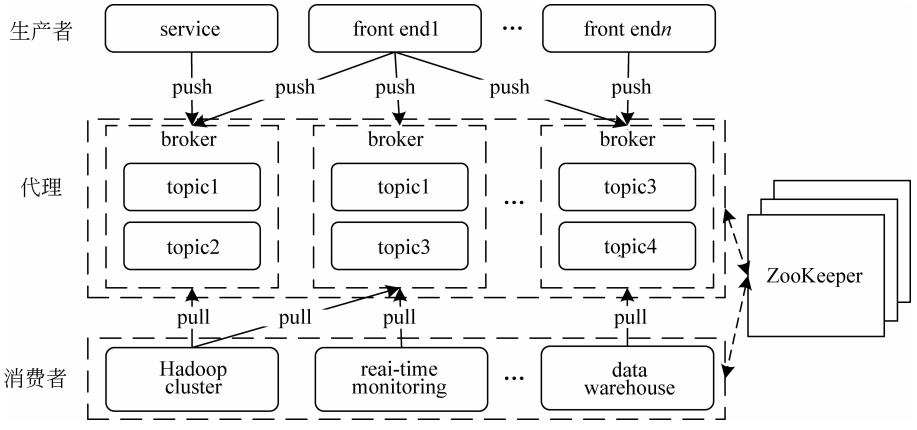

图 3.3　Apache Kafka 系统架构

推送消息，消费者以组为单位，可以关注并读取自己感兴趣的消息。Kafka 通过 ZooKeeper 实现了对生产者和代理的全局状态信息的管理及其负载均衡。

3.2.2 消息、主题和 Schema

在 Kafka 的定义中，消息是一个数据单元，主题是一个数据流的类别或者名称。如果把 Kafka 看作一个关系数据库，那么消息可以看成是数据库里的一个数据行或一条记录。相应地，一个主题就可以理解为数据库中的一张表，主题的名称就是表名。与大多关系数据库不同的是，在 Kafka 中消息并没有字段类型，仅被当作字节数组进行处理和保存。除了被表示成字节数组的数据，消息还可以有一个可选的元信息，称为主键。对于 Kafka 来说，主键也是字节数组，用来控制 Kafka 数据写入不同分区的过程。最简单的情况就是为主键生成一个连续的散列值，根据散列值对分区总数取模进行分区选择，这样可以保证具有同样主键的消息被写入相同的分区中。

为了提升数据写入的效率，Kafka 采用批量写入方式。一个批次就是一组消息集合，这些消息会被写入一个主题和分区。批次写入的方式减少了大量网络 I/O 开销，但是需要在时间延迟和吞吐量之间做出权衡。一个批次中数据量越大，单位时间内处理的消息就越多，单个消息的传输时间就越长。一般一个批次中的数据会被压缩，进一步节省了数据传输和存储的成本，但是需要在解压中耗费一定的计算量。

Kafka 中的消息格式为没有语义的字节数组，因此建议使用 Schema 描述消息的内容，让消息更容易理解。根据应用需求，Schema 有许多可以选择的方式，例如 JSON (JavaScript Object Notation)和 XML 格式，简单易用，可读性很好。不过 JSON 和 XML 缺乏强类型的识别，不同版本之间的兼容性不高。在 Kafka 社区，Apache Avro 是非常受欢迎的一种序列化框架。Apache Avro 最初是为 Hadoop 设计的，提供了一种紧凑的序列化格式，Schema 和消息数据是解耦的，当 Schema 发生变化后，不需要重新格式化数据。同时，Apache Avro 还支持强类型，发行版本也进行了兼容。对于 Kafka 开发，保持 Schema 一致的重要性不言而喻，因为它保证了在写入数据和读取数据时不会因格式不同而造成冲突。

3.2.3 分区

每一个分区(partition)是一个有序列表，写入的数据会按照顺序排列，其中的每一个元素都按照顺序被标记上了 id，称为偏移量(offset)。不同于其他消息中间件，Kafka 中的消息即使被消费了，消息也不会被立即删除。日志文件将会根据 broker 中的配置要求，保留一定的时间之后再删除。如 log 文件保留两天，两天后文件会被清除，无论其中的消息是否被消费。Kafka 通过这种简单的手段释放磁盘空间，以及减少消息消费之后对文件内容改动的磁盘开支。

分区的目的有多个，最根本原因是 Kafka 基于文件存储。通过分区，可以将日志内容分散到多个磁盘上，避免文件尺寸达到单机磁盘的上限，每个分区都会被当前服务器(Kafka 实例)保存；可以将一个主题切分成任意多个分区，提高消息保存和消费的效率，如图 3.4 所示。

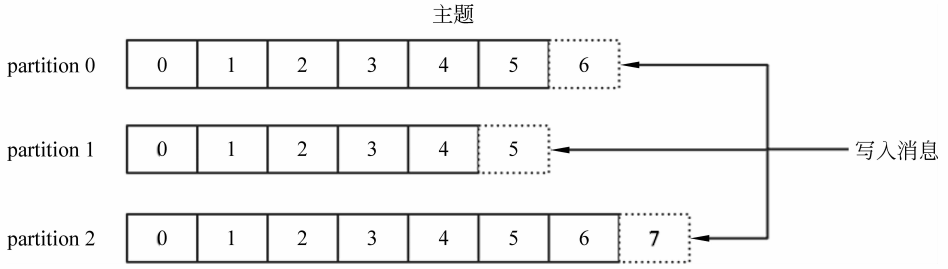

图 3.4　Kafka 的主题

3.2.4　生产者与消费者

生产者创建消息，并把消息发布到一个或多个特定的主题上。一般情况下，生产者默认把消息均衡地分布到主题的所有分区上，而并不关心特定消息会被写到哪个分区。不过生产者也可以把消息直接写到指定的分区。这通常是通过消息键和分区器实现的，分区器为键生成一个散列值，并将其映射到指定的分区上。这样可以保证包含同一个键的消息会被写到同一个分区上。生产者也可以使用自定义的分区器，根据不同的业务规则将消息映射到分区。

消费者读取数据，可以订阅一个或多个主题，并按照消息生成的顺序读取它们。消费者通过检查消息的偏移量区分已经读取过的消息。偏移量是另一种元数据，它是一个不断递增的整数值，在创建消息时，Kafka 会把它添加到消息里。在给定的分区里，每个消息的偏移量都是唯一的。消费者把每个分区最后读取的消息偏移量保存在 ZooKeeper 或 Kafka 上，如果消费者关闭或者重启，它的读取状态不会丢失。

图 3.5 为 Kafka 生产者和消费者。

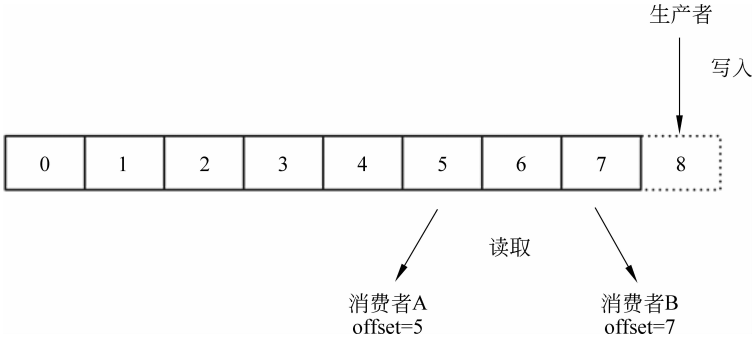

图 3.5　Kafka 生产者和消费者

在计算广告系统中，保存广告的点击信息就是一个典型应用场景。在这个场景里，用户和网站的交互数据会通过生产者实时写入 Kafka 中。生产者不需要关心 Kafka 的数据多久之后会被读取出来用于进行数据分析，它只需要保证每次用户的点击行为都被正确记录。图 3.6 展示了向 Kafka 发送消息的主要步骤。

首先，创建一个 ProducerRecord 对象，该对象需要包含目标主题和要发送的内容，键

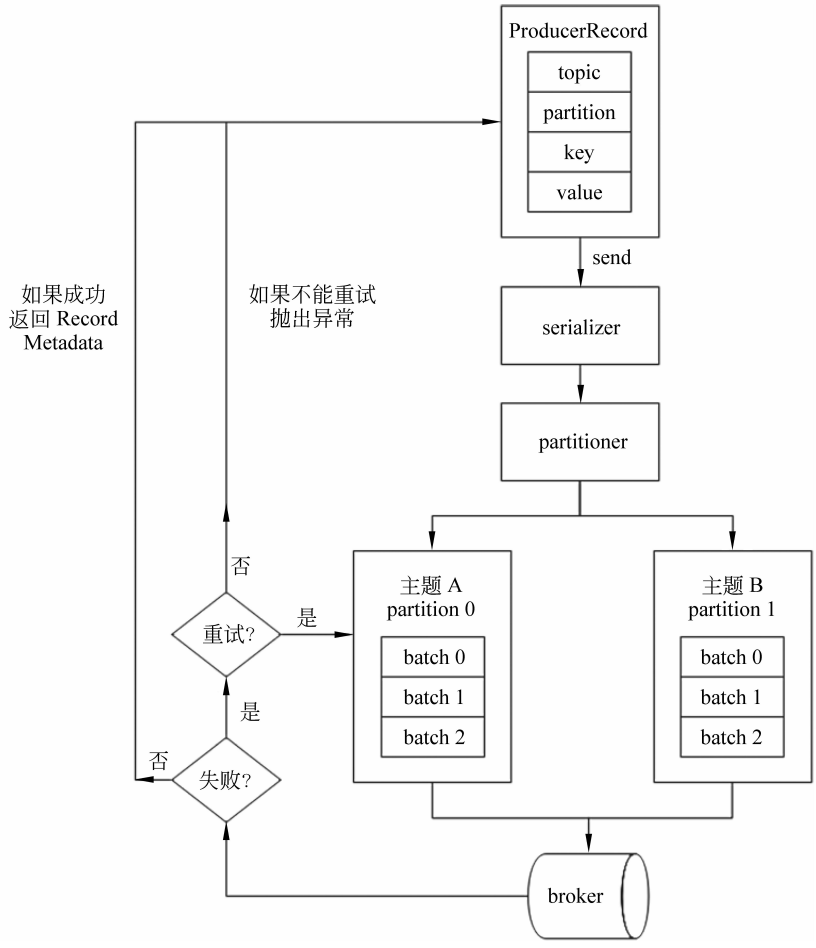

图 3.6　Kafka 数据写入流程

或分区作为可选内容。生产者在发送 ProducerRecord 对象时，要先把对象序列化成字节数组，这样它们才能够在网络上传输。

接下来，数据被传给分区器。如果之前在 ProducerRecord 对象里指定了分区，那么分区器就直接返回之前指定的分区；如果之前没有指定分区，那么分区器会根据 ProducerRecord 对象的键选择一个分区。选好分区以后，生产者就知道该往哪个主题和分区发送这条记录了。紧接着，这条记录被添加到一个记录批次里，这个批次里的所有消息会被发送到相同的主题和分区上。有一个独立的线程负责把这些记录批次发送到相应的代理上。

服务器在收到这些消息时会进行响应：如果消息成功写入 Kafka 中，服务器就返回一个 RecordMetadata 对象，它包含了主题和分区信息，以及记录在分区里的偏移量；如果写入失败，服务器则会返回一个错误信息。生产者在收到错误信息后会尝试重新发送消息，几次之后如果还是失败（次数为生产者配置中的 retries 值），就返回错误信息。

多个消费者可以组成一个消费群组，也就是说，会有一个或多个消费者共同读取一个

主题。消费群组保证每个分区只能被一个消费者使用。通过这种方式，消费者可以消费包含大量消息的主题。而且，如果一个消费者失效，消费群组里的其他消费者可以接管失效消费者的工作，如图 3.7 所示。

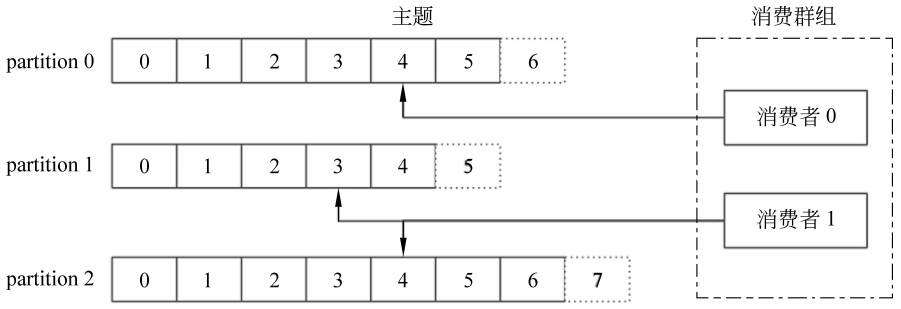

图 3.7　Kafka 消费者与消费群组

对于消费者而言，它需要保存消费消息的偏移量，该偏移量的保存和使用由消费者完全控制；当消费者正常消费消息时，偏移量将会向前驱动，即消息将按照顺序依次被消费。事实上，消费者可以使用任意顺序消费消息，它只需要将偏移量设置为任意值。

Kafka 集群几乎不需要维护任何消费者和生产者的状态信息，这些信息由 ZooKeeper 保存。因此，生产者和消费者的实现非常轻量级，它们可以随意离开，而不会对集群造成额外的影响。

应用程序从 Kafka 中读取消息的时候需要使用 Kafka 消费者进行主题订阅。假设有一个应用程序需要从一个 Kafka 主题读取消息并验证这些消息，然后把它们保存起来。应用程序需要创建一个消费者对象，订阅主题并开始接收消息，然后验证消息并保存结果。过了一段时间，假如生产者往主题写入消息的速度超过了应用程序验证数据的速度，这时候就需要对消费者进行横向扩展。就像多个生产者可以向相同的主题写入消息一样，也可以使用多个消费者从同一个主题读取消息，对消息进行分流。

Kafka 消费者从属于消费群组。一个消费群组里的消费者订阅的是同一个主题，每个消费者接收主题一部分分区的消息。假设主题 T1 有 4 个分区，创建了消费者 C1，它是消费群组 G1 里唯一的消费者，用它订阅主题 T1。消费者 C1 将收到主题 T1 全部 4 个分区的消息，如图 3.8 所示。

如果在消费群组 G1 里新增一个消费者 C2，那么每个消费者将分别从两个分区接收消息。假设消费者 C1 接收分区 0(partition 0)和分区 2(partition 2)的消息，消费者 C2 接收分区 1(partition 1)和分区 3(partition 3)的消息，如图 3.9 所示。

如果消费群组 G1 有 4 个消费者，那么每个消费者可以分配到一个分区，如图 3.10 所示。

如果往消费群组里添加更多的消费者，超过主题的分区数量，那么有一部分消费者就会被闲置，不会接收到任何消息，如图 3.11 所示。

在消费群组里增加消费者是横向扩展消费能力的主要方式。Kafka 的消费者经常会做一些高延迟的操作，如把数据写到数据库或 HDFS 上，或者使用数据进行比较耗时的

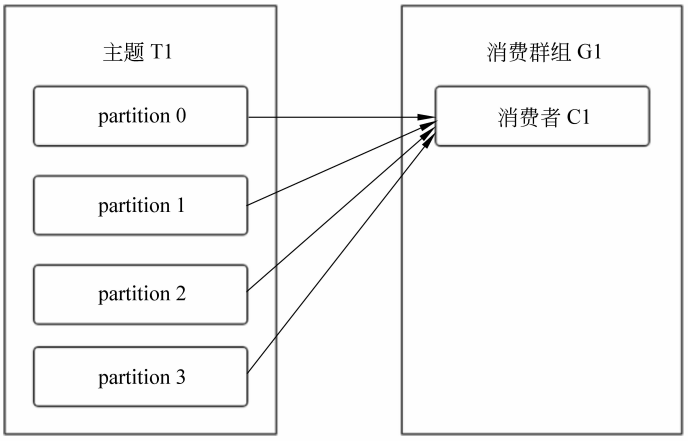

图 3.8　一个消费者

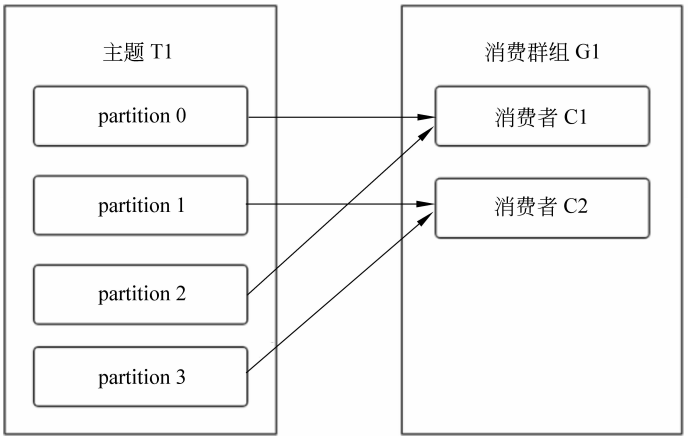

图 3.9　两个消费者

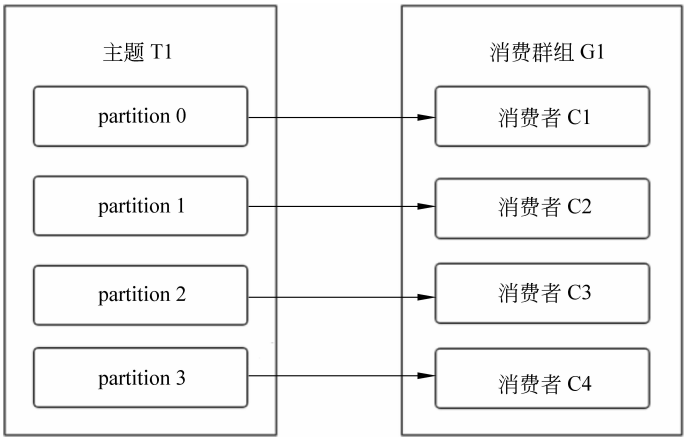

图 3.10　4 个消费者

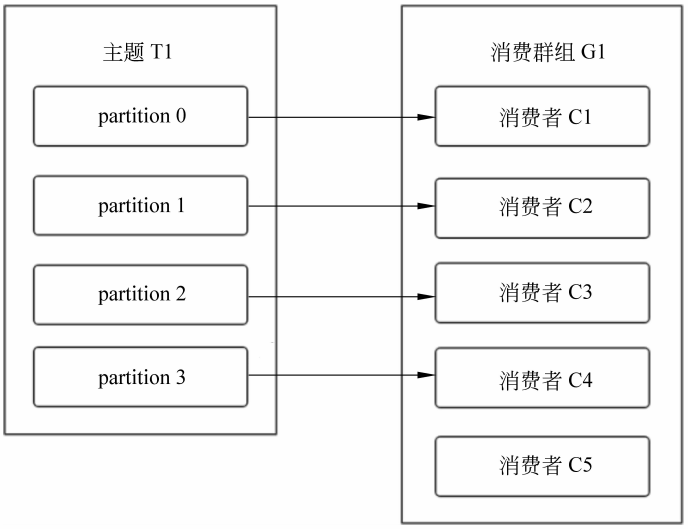

图 3.11　更多的消费者

计算。在这些情况下，单个消费者无法跟上数据生成的速度，所以可以增加更多的消费者，让它们分担负载，每个消费者只处理部分分区的消息，这就是横向扩展的主要手段。为主题创建大量的分区非常必要，在负载增长时可以加入更多的消费者。不过要注意的是，不要让消费者的数量超过主题分区的数量，多余的消费者只会被闲置。

除了通过增加消费者来横向伸缩单个应用程序外，还经常出现多个应用程序从同一个主题读取数据的情况。实际上，Kafka 设计的主要目标之一就是要让 Kafka 主题里的数据能够满足企业各种应用场景的需求。在这些场景里，每个应用程序可以获取到所有的消息，而不只是其中的一部分。只要保证每个应用程序有自己的消费群组，就可以让它们获取到主题所有的消息。不同于传统的消息系统，横向扩展 Kafka 消费者和消费群组并不会对性能造成负面影响。

在上面几个例子里，如果新增一个只包含一个消费者的消费群组 G2，那么这个消费者将从主题 T1 上接收所有的消息，与群组 G1 之间互不影响。消费群组 G2 可以增加更多的消费者，每个消费者可以消费若干个分区，就像消费群组 G1 那样。总的来说，消费群组 G2 还是会接收到所有消息，不管有没有其他群组存在。

简而言之，为每一个需要获取一个或多个主题全部消息的应用程序创建一个消费群组，然后往群组里添加消费者扩展读取能力和处理能力，消费群组里的每个消费者只处理一部分消息。

3.2.5　代理

一个 Kafka 服务称为一个代理(broker)，它同时为生产者和消费者提供服务。

- broker 接收生产者的写入数据请求，接收消息并为消息设置偏移量后把消息保存到磁盘中。
- broker 接收消费者读取分区的请求，返回已经存储的消息给消费者。

Kafka 在设计之初就考虑将多个 broker 组合起来作为一个集群使用。在每个集群中，都有一个 broker 充当集群控制器的角色，这个 broker 是自动从集群中的活跃成员中选举出来的。作为控制器的 broker 负责集群的管理工作，包括将分区分配给其他 broker 以及监控其他 broker 的工作状态。

在集群中，一个分区从属于一个 broker，这个 broker 称为该分区的首领(leader)。但分区可以分配给多个 broker，这样就完成了分区数据的复制。这种复制机制为分区提供了数据冗余，如果有一个 broker 失效，其他 broker 可以接管失效 broker 下的分区。不过，相关的生产者和消费者都要重新连接到新的首领。

相比其他消息中间件，Apache Kafka 的一个重要特性是可以持久化一段时间的信息。Kafka broker 可以为主题配置一个默认的保留时间(例如，7 天)或者一个默认的存储空间(例如，1GB 空间)。当消息保留时间超过 7 天或者消息存储的消息超过 1GB 空间时，旧消息就会过期并被删除，所以在任意时刻，可用消息的总量都不会超过配置参数所指定的大小。每一个主题都可以设置自己的保留策略，如跟踪用户活动的数据可能需要保留几天，而程序指标只需要保留几小时。一些特殊场景，主题可以设定为根据 key 只保留最后一个消息。

3.2.6 Kafka 关键特性

1. 持久化

Kafka 根据设置的保留规则进行数据保存，同时每个主题可以单独设置保留规则，因此 Kafka 可以满足不同消费者的使用需求。消费者不需要担心因为处理速度过慢或遇到流量高峰而导致无法及时读取消息。即使消费者关闭链接，消息仍然会继续保留在 Kafka 中，消费者可以从上次中断的地方继续处理消息。

2. 扩展性

为了能够轻松地处理大数据，Kafka 从一开始就被设计成一个具有灵活伸缩性的系统。用户在开发阶段可以先使用单个 broker，再扩展到包含 3 个 broker 的小型开发集群，然后随着数据量的不断增长，部署到生产环境的集群可能包含上百个 broker。对在线集群进行扩展丝毫不影响整体系统的可用性。也就是说，一个包含多个 broker 的集群，即使个别 broker 失效，仍然可以持续地为客户提供服务。要提高集群的容错能力，需要配置较高的复制系数。

3. 高性能

通过横向扩展生产者、消费者和代理，Kafka 可以轻松地处理巨大的信息流。在处理大量数据的同时，它还能保证亚秒级的消息延迟。

4. 分区再均衡

分区的所有权从一个消费者转移到另一个消费者，这样的行为称为再均衡。再均衡

非常重要，它为消费群组带来了高可用性和扩展性。在再均衡期间，消费者无法读取消息，造成整个消费群组一小段时间的不可用。另外，当分区被重新分配给另一个消费者时，消费者当前的读取状态会丢失，它有可能还需要去刷新缓存，在它重新恢复状态之前会拖慢应用程序。

消费者通过向被指派为消费群组协调器的 broker 发送心跳来维持它们和消费群组的从属关系以及它们对分区的所有权关系。只要消费者以正常的时间间隔发送心跳，就被认为是活跃的，说明它还在读取分区里的消息。消费者会在轮询消息或提交偏移量时发送心跳。如果消费者停止发送心跳的时间足够长，会话就会过期，消费群组协调器认为它已经死亡，就会触发一次再均衡。

5. 提交和偏移量

每次调用 poll 方法，它总是返回生产者写入 Kafka，但是还没有被消费者读取过的记录，因此可以追踪到哪些记录是被消费群组里的哪个消费者读取的。Kafka 不会像其他 JMS 队列那样需要得到消费者的确认，相反，消费者可以使用 Kafka 追踪消息在分区里的位置，或者说是偏移量。

消费者往一个叫作_consumer_offset 的特殊主题发送消息，消息里包含每个分区的偏移量。如果消费者一直处于运行状态，那么偏移量就没有什么用处。不过，如果消费者发生崩溃或者有新的消费者加入消费群组，就会触发再均衡，完成再均衡之后，每个消费者可能分配到新的分区，而不是之前处理的那个分区。为了能够继续之前的工作，消费者需要读取每个分区最后一次提交的偏移量，然后从偏移量指定的地方继续处理。

如果提交的偏移量小于客户端处理的最后一个消息的偏移量，那么处于两个偏移量之间的消息将会被重复处理；如果提交的偏移量大于客户端处理的最后一个消息的偏移量，那么处于两个偏移量之间的消息将会丢失。

6. 复制

复制功能是 Kafka 架构的核心。在 Kafka 的文档里，Kafka 把自己描述成“一个分布式的、可分区的、可复制的提交日志服务”。复制的关键之处在于如果个别节点失效仍能保证 Kafka 的可用性和持久性。

Kafka 使用主题来组织数据，每个主题被分为若干个分区，每个分区有多个副本。这些副本被保存在 broker 上，每个 broker 可以保存成百上千个属于不同主题和分区的副本。

副本有以下两种类型。

(1) leader 副本：每个分区都有一个 leader 副本。为了保证一致性，所有生产者请求和消费者请求多会经过这个副本。

(2) follower 副本：leader 以外的副本都是 follower 副本。follower 副本不处理来自客户端的请求，它们唯一的任务就是从 leader 那里复制消息，保持与 leader 一致的状态。如果 leader 发生崩溃，其中一个 follower 会被提升为新 leader。

leader 的另一个任务是搞清楚哪个 follower 的状态与自己是一致的。follower 为了

保持与 leader 的状态一致，在有新消息到达时尝试从 leader 那里复制消息，不过有各种原因会导致同步失败。例如，网络拥塞导致复制变慢，broker 发生崩溃导致复制滞后，直到重启 broker 后复制才会继续。

为了与 leader 保持同步，follower 向 leader 发送获取数据请求，这种请求和消费者为了读取消息而发送的请求是一样的。leader 将响应消息发送给 follower。请求消息里包含了 follower 想要获取消息的偏移量，而且这些偏移量总是有序的。

7. 可靠性

ACID 是关系数据库普遍支持的可靠性标准，其中 ACID 表示原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）。关系数据库只有满足 ACID 标准，才能确保应用程序安全。我们知道数据库系统承诺可以做到什么，也知道在不同条件下它们会发生怎样的行为。

Kafka 可以保证分区消息的顺序。如果使用同一个生产者往同一个分区写入消息，而且消息 B 在消息 A 之后写入，那么 Kafka 可以保证消息 B 的偏移量比消息 A 的偏移量大，而且消费者会先读取消息 A 再读取消息 B。

只有当消息被写入分区的所有同步副本时，它才被认为是已提交的。生产者可以选择接收不同类型的确认，如在消息被完全提交时的确认，或者在消息被写入 leader 副本时的确认，或者在消息被发送到网络时的确认。只要还有一个副本是活跃的，那么已经提交的消息就不会丢失。消费者只能读取已经提交的消息。

这些基本的保证机制可以用来构建可靠的系统，但仅仅依赖它们是无法保证系统完全可靠的。构建一个可靠的系统需要做出一些权衡，Kafka 管理员和开发者可以在配置参数上做出权衡，从而得到他们想要达到的可靠性。这种权衡一般是指消息存储的可靠性和一致性的重要程度与可用性、高吞吐量、低延迟和硬件成本的重要程度之间的权衡。

3.2.7 项目实践 4：通过 Kafka 进行数据处理

广告系统产生大量线上展示数据，如果数据直接写入 HDFS，Hadoop Session 无法承受，而且会严重影响 Hadoop 性能。因此系统会通过消息中间件作为缓存。本项目通过 Python 实现 KafkaProducer 和 KafkaConsumer。

本实践中，模拟一个简单的广告业务数据的收集与处理系统。

1. Kafka 管理

1）启动服务

Kafka 使用 ZooKeeper 进行配置管理，因此启动 Kafka Server 之前需要先启动 ZooKeeper Server。命令如下：

```
>bin/zookeeper-server-start.sh config/zookeeper.properties
[2019-03-12 15:11:30,836] INFO Reading configuration from: config/zookeeper
.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
```

如果启动后出现 java. net. BindException：Address already in use 这样的错误，说明 ZooKeeper 要使用的 2181 端口已经被占用。这时可以通过如下命令查看 2181 端口被哪个进程占用：

```
>lsof -i:2181
COMMAND  PID    USER  FD  TYPE  DEVICE SIZE/OFF NODE NAME
java  17177 zookeeper  41u  IPv4  43551758    0t0  TCP * :eforward (LISTEN)
```

从上面的信息可以看到已经有一个 ZooKeeper 在运行了，这是因为启动的 Hadoop 也使用了 ZooKeeper，这样就不需要再次启动 ZooKeeper 了。但是如果该端口号不是被 ZooKeeper 占用，就需要考虑是 kill 占用的进程释放端口号，还是通过修改配置让 ZooKeeper 服务使用其他端口号。ZooKeeper 的端口号配置在 config/zookeeper . properties 中，设置方法在行“clientPort＝2181”。

确认了 ZooKeeper 正常启动后，开始启动 Kafka Server：

```
>bin/kafka-server-start.sh config/server.properties
[2019-03-12 15:13:19,621] INFO Registered kafka:type=kafka.Log4jController
MBean (kafka.utils.Log4jControllerRegistration$)
```

注意：如果在启动 ZooKeeper 时修改了端口号，那么也需要在 Kafka Server 的配置文件 config/server. properties 中进行对应修改，设置方法在行“zookeeper. connect＝localhost：2181”。

2）创建 topic

使用 Kafka 的第一件事情就是创建一个 topic，使用如下命令创建一个名字为 test 的 topic：

```
>bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-
factor 1 --partitions 1 --topic test
Created topic "test".
```

之后可以通过下面的命令查看 topic 是否已经成功创建。

```
>bin/kafka-topics.sh --list --zookeeper localhost:2181
test
```

3）发送数据和消费数据

有了 topic 后，就可以使用一个简单的脚本向 Kafka 的 topic 中写入数据，下面的命令启动了一个生产者，等待用户输入。

```
>bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

输入数据：

```
This is a message
This is another message
```

写入数据后，可以启动一个消费者从 topic 中从头读取数据，命令如下：