# 第3章 数据采集

学习目标

- 了解用户行为数据，能够说出电商网站中用户行为数据的含义。
- 了解模拟生成用户行为数据，能够实现模拟生成用户行为数据的 Python 程序。
- 掌握配置采集方案，能够根据需求灵活配置 Flume 的采集方案。
- 熟悉采集用户行为数据，能够根据采集方案启动 Flume 采集数据。

数据采集是指通过各种技术手段从不同数据源获取数据的过程。其目标是获得完整、准确、及时的数据，以支持后续的数据分析和决策。本项目的核心需求是分析电商网站中的用户行为数据，这些数据通过网站的埋点获取，并以日志的形式存储在服务器上。本章详细介绍如何通过数据采集来获取用户行为数据。

## 3.1 用户行为数据概述

用户行为数据是指用户在电商网站上的各种交互记录，包括用户的行为信息及环境信息。收集这些数据的主要目的是优化产品，并为各项分析统计指标提供数据支持。本项目采集的用户行为数据主要包括页面信息、设备信息和行为信息。下面以一条用户行为数据为例进行说明，具体内容如下。

```
{
    "page_info": {
        "page_id": 287,
        "page_url": "https://www.example.com/page_287",
        "product_id": 287,
        "category": "Grocery"
    },
    "behavior_info": {
        "user_id": 6421,
        "behavior_type": "purchase",
        "action_time": "2023-02-15 05:32:31",
        "location": "湖北, 宜昌"
    },
```

```
        "device_info": {
            "operating_system": "Android",
            "access_method": "browser",
            "browser_type": "Opera",
            "app_version": null
        }
    }
```

从上述内容可以看出,本项目所采集的用户行为数据以对象结构的 JSON 格式存储。其中,键 page_info 的值以对象结构存储页面信息,键 behavior_info 的值以对象结构存储行为信息,键 device_info 的值以对象结构存储设备信息。有关这些信息的详细说明如表 3-1 所示。

表 3-1 用户行为数据的详细说明

| 类 别 | 键 | 描 述 |
| --- | --- | --- |
| 页面信息 | page_id | 表示用户所访问页面的唯一标识 |
| | page_url | 表示用户所访问页面的 URL 地址 |
| | product_id | 表示商品的唯一标识 |
| | category | 表示商品所属的品类 |
| 行为信息 | user_id | 表示用户的唯一标识 |
| | behavior_type | 表示用户的行为类型,其值包括 click(点击)、cart(加入购物车)和 purchase(购买) |
| | action_time | 表示用户触发行为的时间 |
| | location | 表示用户触发行为的地理位置 |
| 设备信息 | operating_system | 表示用户使用的操作系统 |
| | access_method | 表示用户的访问方式,其值包括 app 和 browser(浏览器) |
| | browser_type | 表示浏览器类型,当用户的访问方式为 app 时,浏览器类型的值为 null |
| | app_version | 表示 App 的版本号,当用户的访问方式为 browser 时,App 的版本号为 null |

## 3.2 模拟生成用户行为数据

本项目通过编写 Python 程序模拟生成用户行为数据。由于离线分析和实时分析所用的用户行为数据分别来自历史数据和实时数据,它们在用户触发行为的时间上会有差异,所以需要编写两个 Python 程序,以便生成两种不同类型的用户行为数据。本节讲解如何通过编写 Python 程序模拟生成用户行为数据。

### 3.2.1　生成历史用户行为数据

在本项目中，需要生成一年的历史用户行为数据，时间范围是 2023 年 1 月 1 日至 2023 年 12 月 31 日。接下来演示如何使用集成开发工具 PyCharm 编写 Python 程序，实现生成历史用户行为数据的功能，具体操作步骤如下。

#### 1. 创建项目

在 PyCharm 中基于自定义环境创建名为 spark_project 的项目，并指定项目使用本地安装的 Python 3.9.13 版本的 Python 解释器，如图 3-1 所示。

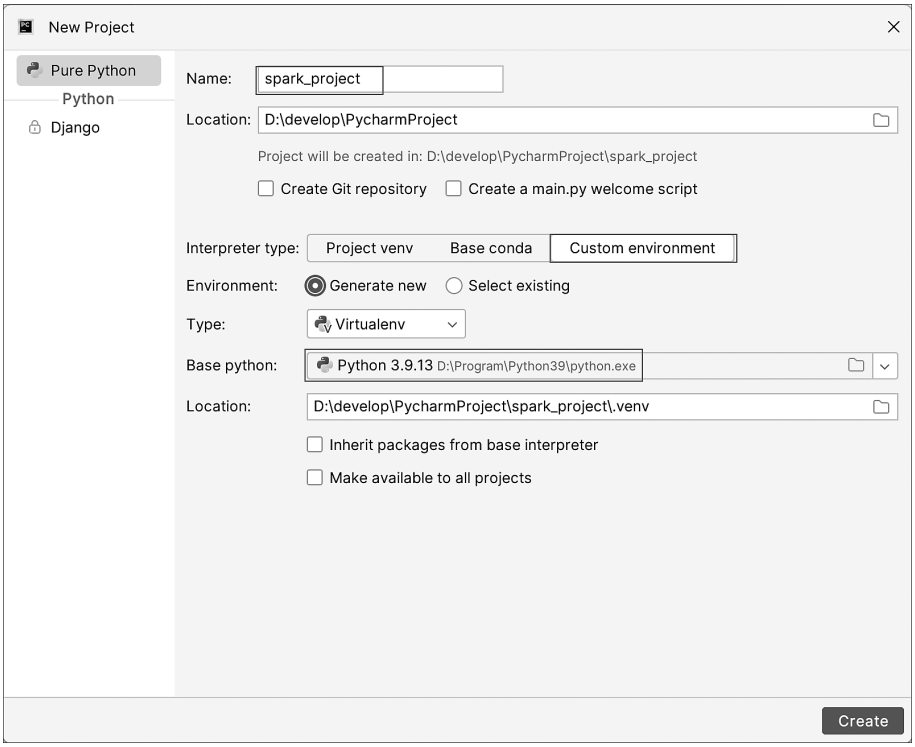


图 3-1　创建项目

在图 3-1 中，单击 Create 按钮创建项目 spark_project。

#### 2. 创建目录

在项目 spark_project 中创建名为 data 的目录，用于存放生成用户行为数据的 Python 文件，如图 3-2 所示。

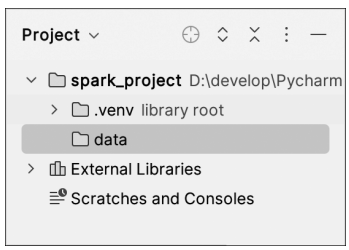


图 3-2　创建目录

#### 3. 创建 Python 文件

在项目 spark_project 的 data 目录中创建名为 generate_user_data_history 的 Python 文件，用于实现生成历史用户行为数据的 Python 程序。

#### 4. 实现 Python 程序

在 generate_user_data_history.py 文件中，添加用

于生成历史用户行为数据的相关模块和代码,具体操作步骤如下。

(1) 在 generate_user_data_history.py 文件中导入 json、random、datetime 和 time 模块,具体代码如文件 3-1 所示。

**文件 3-1 generate_user_data_history.py**

```
#用于处理 JSON 格式的数据
import json
#用于生成随机数
import random
#用于处理日期和时间
import datetime
#用于提供与时间相关的函数
import time
```

(2) 在文件 3-1 中添加名为 random_date 的函数,用于生成一个在指定时间范围内随机的时间作为用户触发行为的时间,具体代码如下。

```
def random_date(start, end):
    return start + datetime.timedelta(
        seconds=random.randint(0, int((end - start).total_seconds())),
    )
```

上述代码中,函数 random_date()接收两个参数 start 和 end,分别用于指定时间范围的起始和结束时间。

(3) 在文件 3-1 中添加名为 random_location 的函数,用于生成用户触发行为的地理位置,具体代码如下。

```
def random_location():
    locations = {
        "北京": ["北京"],
        "上海": ["上海"],
        "广东": ["广州", "深圳", "东莞", "珠海"],
        "浙江": ["杭州", "宁波", "温州", "嘉兴", "湖州"],
        "江苏": ["南京", "苏州", "无锡", "常州", "扬州"],
        "四川": ["成都", "绵阳", "德阳", "南充", "宜宾"],
        "湖北": ["武汉", "黄石", "宜昌", "襄阳", "荆州"],
        "山东": ["济南", "青岛", "烟台", "潍坊", "淄博"],
        "河南": ["郑州", "洛阳", "开封", "新乡", "安阳"],
        "河北": ["石家庄", "唐山", "邯郸", "张家口"],
        "湖南": ["长沙", "株洲", "湘潭", "衡阳", "岳阳"]
    }
    province = random.choice(list(locations.keys()))
    city = random.choice(locations[province])
```

```
17      return f"{province}, {city}"
```

上述代码中，第 2～14 行代码定义了一个字典 locations，其中包含省份及其对应城市的信息。每个省份作为键，对应一个包含该省份城市的列表。第 15 行代码用于随机选择一个省份。第 16 行代码则基于已选择的省份，随机选择该省份的一个城市。

（4）在文件 3-1 中添加名为 generate_page_info 的函数，用于生成页面信息，具体代码如下。

```
1   def generate_page_info():
2       product_categories = {
3           range(1, 31): "Electronics",
4           range(31, 61): "Clothing",
5           range(61, 91): "Books",
6           range(91, 121): "Home",
7           range(121, 151): "Toys",
8           range(151, 181): "Sports",
9           range(181, 211): "Beauty",
10          range(211, 241): "Health",
11          range(241, 271): "Automotive",
12          range(271, 301): "Grocery"
13      }
14      product_id = random.randint(1, 300)
15      category = next(
16          (
17              cat for range_,
18              cat in product_categories.items() if product_id in range_
19          )
20      )
21      page_info = {
22          "page_id": product_id,
23          "page_url": f"https://www.example.com/page_{product_id}",
24          "product_id": product_id,
25          "category": category
26      }
27      return page_info
```

上述代码中，第 2～13 行代码使用字典 product_categories 来定义不同范围商品对应的品类。第 14 行代码生成了一个在指定范围内的随机整数，作为商品的唯一标识，同时也作为页面的唯一标识，以确保每个页面只对应一个商品。第 15～20 行代码用于根据生成的商品的唯一标识确定其所属的品类。第 21～26 行代码定义了一个包含页面信息的字典 page_info，该字典的第一个键值对表示用户所访问页面的唯一标识，第二个键值对表示用户所访问页面的 URL 地址，第三个键值对表示商品的唯一标识，第四个键值对表

示商品所属品类。

(5) 在文件 3-1 中添加名为 generate_device_info 的函数,用于生成设备信息,具体代码如下。

```
def generate_device_info():
    access_method = random.choice(["browser", "app"])
    device_info = {
        "operating_system": random.choice(
            ["Windows", "macOS", "Android", "iOS"]
        ),
        "access_method": access_method
    }
    if access_method == "browser":
        device_info["browser_type"] = random.choice(
            ["Chrome", "Firefox", "Safari", "Edge", "Opera"]
        )
        device_info["app_version"] = None
    elif access_method == "app":
        device_info["browser_type"] = None
        device_info["app_version"] = (f"{random.randint(8, 10)}."
                                      f"{random.randint(0, 9)}."
                                      f"{random.randint(0, 9)}")
    return device_info
```

上述代码中,第 2 行代码用于随机选择一个用户的访问方式。第 3～8 行代码定义了一个包含设备信息的字典 device_info,该字典的第一个键值对表示用户使用的操作系统,第二个键值对表示用户的访问方式。

第 9～18 行代码用于通过判断用户的访问方式,向字典 device_info 中添加两个键值对,它们的键分别为 browser_type 和 app_version,表示浏览器类型和 App 版本号。当用户的访问方式为 browser 时,键 app_version 的值为 None,表示没有 App 版本号的信息。当用户的访问方式为 app 时,键 browser_type 的值为 None,表示没有浏览器类型的信息。

(6) 在文件 3-1 中添加名为 generate_behavior_info 的函数,用于生成行为信息,具体代码如下。

```
def generate_behavior_info():
    start_date = datetime.datetime(2023, 1, 1)
    end_date = datetime.datetime(2023, 12, 31,23,59,59)
    behavior_info = {
        "user_id": random.randint(1, 10000),
        "behavior_type": random.choice(["click", "cart", "purchase"]),
        "action_time": str(random_date(start_date, end_date)),
```

```
        "location": random_location()
    }
    return behavior_info
```

上述代码中，第 2 行代码用于指定时间范围的起始时间为 2023-01-01 00:00:00。第 3 行代码用于指定时间范围的结束时间为 2023-12-31 23:59:59。第 4～9 行代码定义了一个包含行为信息的字典 behavior_info，该字典的第一个键值对表示用户的唯一标识，第二个键值对表示用户的行为类型，第三个键值对表示用户触发行为的时间，第四个键值对表示用户触发行为的地理位置。

（7）在文件 3-1 中添加名为 generate_user_behavior 的函数，用于整合页面信息、行为信息和设备信息，从而生成完整的用户行为数据，具体代码如下。

```
def generate_user_behavior():
    page_info = generate_page_info()
    behavior_info = generate_behavior_info()
    device_info = generate_device_info()
    user_behavior = {
        "page_info": page_info,
        "behavior_info": behavior_info,
        "device_info": device_info
    }
    return user_behavior
```

上述代码中，第 2～4 行代码分别用于获取页面信息、行为信息和设备信息。第 5～9 行代码定义了一个包含用户行为数据的字典 user_behavior，该字典的第一个键值对表示页面信息，第二个键值对表示行为信息，第三个键值对表示设备信息。

（8）在文件 3-1 中添加名为 output_user_behaviors 的函数，用于将生成的用户行为数据写入日志文件，具体代码如下。

```
def output_user_behaviors(interval, output_file):
    try:
        with open(output_file, 'a', encoding='utf-8') as f:
            while True:
                #生成用户行为数据
                user_behavior = generate_user_behavior()
                #将用户行为数据转换为 JSON 格式
          user_behavior_json = json.dumps(user_behavior,ensure_ascii=False)
                f.write(user_behavior_json + "\n")
                f.flush()
                time.sleep(interval)
    except KeyboardInterrupt:
        print("Data generation stopped.")
```

上述代码中,函数 output_user_behaviors()接收两个参数 interval 和 output_file,分别用于指定生成每条用户行为数据的时间间隔(秒),以及日志文件所在目录和名称。

(9) 在文件 3-1 中调用函数 output_user_behaviors(),指定生成每条用户行为数据的时间间隔为 0.5 秒,日志文件所在目录为/export/data/log/2023,日志文件名称为 user_behaviors.log,具体代码如下。

```
output_user_behaviors(0.5,"/export/data/log/2023/user_behaviors.log")
```

需要说明的是,由于本项目将使用虚拟机 Spark03 运行 Python 程序,因此上述代码指定的目录为 Linux 操作系统的格式。

5. 创建目录

在虚拟机 Spark03 中创建用于存储日志文件 user_behaviors.log 的目录/export/data/log/2023,具体命令如下。

```
mkdir -p /export/data/log/2023
```

### 3.2.2 生成实时用户行为数据

在本项目中,生成实时和历史用户行为数据的 Python 程序基本一致,不同之处在于,实时用户行为数据使用系统当前时间来生成用户触发行为的时间。因此,可以参考 generate_user_data_history.py 文件中的代码,来编写生成实时用户行为数据的 Python 程序,操作步骤如下。

1. 创建 Python 文件

在项目 spark_project 的 data 目录中创建名为 generate_user_data_real 的 Python 文件,用于实现生成实时用户行为数据的 Python 程序。

2. 实现 Python 程序

在 generate_user_data_real.py 文件中,添加用于生成实时用户行为数据的相关模块和代码,具体操作步骤如下。

(1) 将 generate_user_data_history.py 文件中的内容复制到 generate_user_data_real.py 文件中。

(2) 在 generate_user_data_real.py 文件中,将导入 datetime 模块的代码替换为如下代码。

```
from datetime import datetime
```

(3) 在 generate_user_data_real.py 文件中导入 os 模块用于与操作系统交互,具体代码如下。

```
import os
```

(4) 在 generate_user_data_real.py 文件中,删除名为 random_date 的函数。

(5) 在 generate_user_data_real.py 文件中，修改名为 generate_behavior_info 的函数，该函数修改完成的内容如下。

```
def generate_behavior_info():
    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    behavior_info = {
        "user_id": random.randint(1, 10000),
        "behavior_type": random.choice(["click", "cart", "purchase"]),
        "action_time": current_time,
        "location": random_location()
    }
    return behavior_info
```

上述代码中，第 2 行代码用于获取当前系统时间，并将其转换为 YYYY-mm-dd HH:mm:ss(年-月-日 时:分:秒)格式的字符串。

(6) 在 generate_user_data_real.py 文件中，将调用函数 output_user_behaviors()的代码修改为如下内容。

```
#根据当前系统时间获取年份
current_year = datetime.now().year
#指定日志文件所在目录
directory_template = "/export/data/log/{year}"
#定义日志文件的名称
file_name = "user_behaviors.log"
#将指定目录中的占位符替换为获取的年份
directory = directory_template.format(year=current_year)
#判断目录是否存在,若不存在,则创建该目录
os.makedirs(directory, exist_ok=True)
#将目录和日志文件的名称合并成一个完整的文件路径
output_file = os.path.join(directory, file_name)
output_user_behaviors(0.5,output_file)
```

上述代码通过调用函数 output_user_behaviors()将用户行为数据写入指定目录的日志文件 user_behaviors.log。其中，/export/data/log 目录的子目录会根据当前系统时间中的年份自动生成。

## 3.3　配置采集方案

本项目需要在虚拟机 Spark03 上启动两个 Flume Agent，分别负责采集历史和实时用户行为数据。因此，需要为这两个 Flume Agent 配置不同的采集方案，以适应不同的数据采集需求，具体实现过程如下。

### 1. 配置采集历史用户行为数据的方案

采集历史用户行为数据的 Flume Agent 在启动时，会对数据进行 JSON 格式校验，

以确保后续数据分析和存储过程中使用的数据符合 JSON 格式要求。校验通过后，Flume Agent 将根据用户行为触发时间的日期，将历史用户行为数据发送到 HDFS 的不同目录中。

采集历史用户行为数据的 Flume Agent 主要包括 Source、Channel 和 Sink 三个组件。其中，Source 组件的类型为 Taildir Source，负责监控和读取日志文件 user_behaviors.log 中的历史用户行为数据。Channel 组件的类型为 File Channel，负责将 Flume 中的事件持久化到磁盘上。Sink 组件的类型为 HDFS Sink，负责将历史用户行为数据输出到 HDFS 的指定目录。

接下来演示如何在虚拟机 Spark03 中配置采集历史用户行为数据的方案，操作步骤如下。

(1) 在虚拟机 Spark03 中创建/export/data/flume_conf 目录，用于存放采集方案的配置文件，具体命令如下。

```
mkdir /export/data/flume_conf
```

(2) 在虚拟机 Spark03 的/export/data/flume_conf 目录中，使用 vi 编辑器编辑配置文件 flume-logs-history.conf，在该文件中添加采集方案，具体内容如文件 3-2 所示。

**文件 3-2　flume-logs-history.conf**

```
#定义 Source 组件的标识 r1
a1.sources = r1
#定义 Channel 组件的标识 c1
a1.channels = c1
#定义 Sink 组件的标识 k1
a1.sinks = k1
#定义 Source 组件的类型为 Taildir Source
a1.sources.r1.type = TAILDIR
#定义用于记录被监控文件当前读取位置的文件 taildir_position_history.json
a1.sources.r1.positionFile = /export/data/flume/taildir_position_
history.json
#定义文件组的标识为 f1
a1.sources.r1.filegroups = f1
#定义文件组 f1 中被监控文件的位置，即日志文件 user_behaviors.log 所在目录
a1.sources.r1.filegroups.f1 = /export/data/log/2023/user_behaviors.log
#定义 Source 组件中拦截器的标识 i1
a1.sources.r1.interceptors = i1
#在标识为 i1 的拦截器中添加一个自定义拦截器，用于校验数据是否为 JSON 格式并将用
#户触发行为的时间转换为时间戳格式之后添加到事件的 header 中
a1.sources.r1.interceptors.i1.type = cn.itcast.flume
.JsonAndTimestampInterceptor$Builder
#定义 Channel 组件的类型为 File Channel
a1.channels.c1.type = file
```

```
#定义 File Channel 存储元数据的目录
a1.channels.c1.checkpointDir = /export/data/flume/checkpoint
#定义 File Channel 存储事件的目录
a1.channels.c1.dataDirs = /export/data/flume/data
#定义 Sink 组件的类型为 HDFS Sink
a1.sinks.k1.type = hdfs
#定义 HDFS Sink 将数据输出到指定目录的文件中,其中%Y-%m-%d 表示根据日期创建目
#录,如 2023-11-02
a1.sinks.k1.hdfs.path = /origin_data/log/user_behaviors/%Y-%m-%d
#定义文件的前缀为 log
a1.sinks.k1.hdfs.filePrefix = log
#定义滚动新文件的时间间隔为 10 秒
a1.sinks.k1.hdfs.rollInterval = 10
#定义滚动新文件的大小为 0,表示不根据文件大小滚动新文件
a1.sinks.k1.hdfs.rollSize = 0
#定义滚动新文件的事件数为 0,表示不根据事件数滚动新文件
a1.sinks.k1.hdfs.rollCount = 0
#定义文件的类型为压缩文件,以减少存储空间和提高传输效率
a1.sinks.k1.hdfs.fileType = CompressedStream
#定义压缩文件的压缩编解码器为 GZIP
a1.sinks.k1.hdfs.codeC = gzip
#将 Source 组件与 Channel 组件关联
a1.sources.r1.channels = c1
#将 Sink 组件与 Channel 组件关联
a1.sinks.k1.channel = c1
```

在文件 3-2 中,指定 Flume Agent 的标识为 a1,其中第 28 行代码依据的日期来源于每条用户行为数据中用户触发行为的时间。第 18 行代码添加的自定义拦截器需要通过编写 Java 程序来实现,其具体实现过程本书不作重点讲解。在本书的配套资源中提供了自定义拦截器的 jar 文件 FlumeInterceptor.jar,供读者直接使用。第 40 行代码使用压缩编解码器 GZIP 是因为其具有较高的压缩率,可以最大程度减少存储空间。

在文件 3-2 中添加采集方案后,保存并退出编辑。

### 2. 配置采集实时用户行为数据的方案

采集实时用户行为数据的 Flume Agent 在启动时,会对数据进行 JSON 格式校验,以确保后续数据分析和存储过程中使用的数据符合 JSON 格式要求。校验通过后,Flume Agent 将实时用户行为数据发送到 Kafka。

采集实时用户行为数据的 Flume Agent 主要包括 Source 和 Channel 两个组件。其中,Source 组件的类型为 Taildir Source,负责监控和读取日志文件 user_behaviors.log 中的用户行为数据。Channel 组件的类型为 Kafka Channel,用于将用户行为数据传输到 Kafka。

在虚拟机 Spark03 的/export/data/flume_conf 目录中,使用 vi 编辑器编辑配置文件

flume-logs-real.conf,在该文件中添加采集实时用户行为数据的方案,具体内容如文件 3-3 所示。

**文件 3-3 flume-logs-real.conf**

```
#定义 Source 组件的标识 r1
a2.sources = r1
#定义 Channel 组件的标识 c1
a2.channels = c1
#定义 Source 组件的类型为 Taildir Source
a2.sources.r1.type = TAILDIR
#定义用于记录被监控文件当前读取位置的文件 taildir_position_real.json
a2.sources.r1.positionFile = /export/data/flume/taildir_position_real
.json
#定义文件组的标识为 f1
a2.sources.r1.filegroups = f1
#定义文件组 f1 中被监控文件的位置,即日志文件 user_behaviors.log 所在目录
a2.sources.r1.filegroups.f1 = /export/data/log/2024/user_behaviors.log
#定义 Source 组件中拦截器的标识 i1
a2.sources.r1.interceptors = i1
#在标识为 i1 的拦截器中添加一个自定义拦截器,用于校验数据是否为 JSON 格式
a2.sources.r1.interceptors.i1.type = cn.itcast.flume
.JsonValidationInterceptor$Builder
#定义 Channel 组件的类型为 Kafka Channel
a2.channels.c1.type = org.apache.flume.channel.kafka.KafkaChannel
#定义 Kafka 集群的地址
a2.channels.c1.kafka.bootstrap.servers = spark01:9092,spark02:9092,spark03:
9092
#定义 Kakfa 的主题 user_behavior_topic
a2.channels.c1.kafka.topic = user_behavior_topic
#关闭 Flume 对数据的事件解析功能
a2.channels.c1.parseAsFlumeEvent = false
#将 Channel 组件与 Source 组件关联
a2.sources.r1.channels = c1
```

在文件 3-3 中指定 Flume Agent 的标识为 a2。第 16 行代码添加的自定义拦截器需要通过编写 Java 程序来实现,其具体实现过程本书不作重点讲解。在本书的配套资源中提供了自定义拦截器的 jar 文件 FlumeInterceptor.jar,供读者直接使用。

在文件 3-3 中添加采集方案后,保存并退出编辑。需要注意的是,文件 3-3 中日志文件 user_behaviors.log 所在目录需要根据 generate_user_data_real.py 文件运行时实际生成的目录进行修改。

### 3. 添加自定义拦截器

参考第 2 章上传 JDK 安装包的方式,将 jar 文件 FlumeInterceptor.jar.jar 上传到虚拟机 Spark03 的/export/servers/flume-1.10.1/lib 目录中,从而在 Flume 中添加自定义

拦截器。

## 3.4　采集用户行为数据

本节讲解如何使用 3.3 节配置的采集方案，分别启动负责采集历史和实时用户行为数据 Flume Agent，以完成本项目中采集用户行为数据的功能，具体内容如下。

### 1. 启动采集历史用户行为数据的 Flume Agent

在虚拟机 Spark03 中启动 Flume Agent 的操作步骤如下。

（1）启动 HDFS 集群。确保虚拟机 Spark01、Spark02 和 Spark03 中 HDFS 集群的相关进程正常启动。需要说明的是，为了优化资源利用，在采集历史用户行为数据时，可以选择仅启动集群环境中的 HDFS 集群。

（2）参考第 2 章上传 JDK 安装包的方式，将 Python 文件 generate_user_data_history.py 上传到虚拟机 Spark03 的/export/servers 目录中。

（3）在虚拟机 Spark03 中启动 Flume Agent，从/export/data/log/2023 目录中的日志文件 user_behaviors.log 里采集历史用户行为数据，具体命令如下。

```
flume-ng agent --name a1 --conf conf/ --conf-file \
/export/data/flume_conf/flume-logs-history.conf \
-Dflume.root.logger=INFO,console
```

上述命令中，参数--name 指定的参数值 a1 为 Flume Agent 的标识，该标识需要与配置文件 flume-logs-history.conf 中 Flume Agent 的标识一致。

上述命令执行完成后，Flume Agent 会占用 Tabby 中虚拟机 Spark03 的操作窗口，因此用户无法进行其他操作。读者可以在 Tabby 中右击虚拟机 Spark03 的操作窗口，在弹出的菜单中选择“克隆”选项，通过克隆的方式创建一个虚拟机 Spark03 的新操作窗口，如图 3-3 所示。

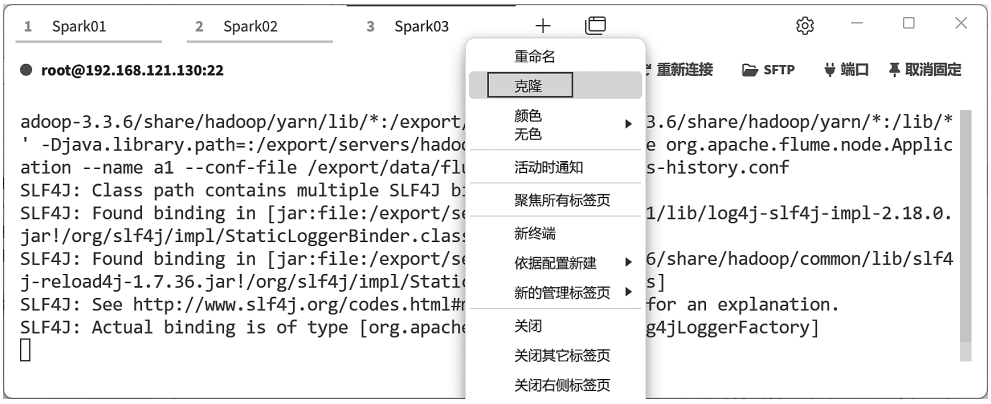


图 3-3　创建一个虚拟机 Spark03 的新操作窗口

（4）在 Tabby 中创建一个虚拟机 Spark03 的新操作窗口，用于执行 Python 文件

generate_user_data_history.py，向/export/data/log/2023 目录的日志文件 user_behaviors.log 中写入历史用户行为数据，具体命令如下。

```
python /export/servers/generate_user_data_history.py
```

上述命令执行完成后，Python 程序会占用 Tabby 中虚拟机 Spark03 的操作窗口，因此用户无法进行其他操作。

(5) 在 HDFS 的/origin_data/log/user_behaviors 目录中，检查采集的历史用户行为数据是否根据用户触发行为时间中的日期，正确分配到 HDFS 的不同目录中。在虚拟机 Spark01 执行如下命令。

```
hdfs dfs -ls /origin_data/log/user_behaviors
```

上述命令执行完成的效果如图 3-4 所示。

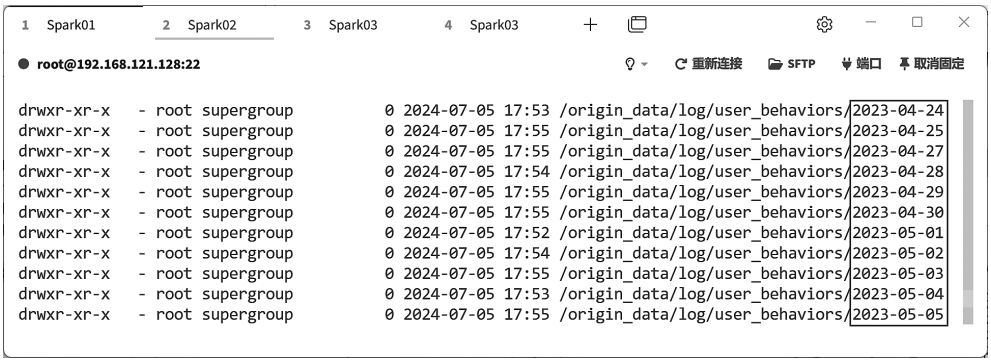


图 3-4 查看/origin_data/log/user_behaviors 目录中的内容

图 3-4 展示了/origin_data/log/user_behaviors 目录的部分内容。该目录下有许多按照日期命名的子目录，每个子目录中包含了对应日期的历史用户行为数据。

(6) 查看/origin_data/log/user_behaviors 目录中任意子目录包含的文件。这里以/origin_data/log/user_behaviors/2023-04-29 目录为例，在虚拟机 Spark01 执行如下命令。

```
hdfs dfs -ls /origin_data/log/user_behaviors/2023-04-29
```

上述命令执行完成的效果如图 3-5 所示。

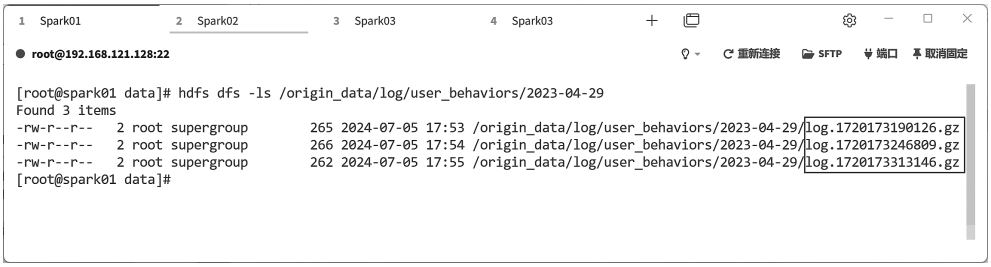


图 3-5 查看/origin_data/log/user_behaviors/2023-04-29 目录的内容

从图 3-5 可以看出，/origin_data/log/user_behaviors/2023-04-29 目录下有 3 个压缩文件(.gz)，这些压缩文件存储了 2023 年 4 月 29 日的用户行为数据。需要说明的是，读者在实际操作时，图 3-5 显示的文件名会与此不同。

(7) 查看/origin_data/log/user_behaviors/2023-04-29 目录中任意压缩文件包含的内容。这里以 log.1720173246809.gz 文件为例，在虚拟机 Spark01 执行如下命令。

```
hdfs dfs -text \
/origin_data/log/user_behaviors/2023-04-29/log.1720173246809.gz
```

上述命令执行完成的效果如图 3-6 所示。


图 3-6　查看 log.1720173246809.gz 文件的内容

从图 3-6 可以看出，log.1720173246809.gz 文件中包含一条用户行为数据，该数据中用户触发行为的时间为 2023 年 4 月 29 日。因此说明，采集的用户行为数据根据用户触发行为时间中的日期，正确分配到 HDFS 的不同目录中。

小提示：虚拟机 Spark03 中 Flume Agent 和 Python 程序的运行时长决定了生成历史用户行为数据的数量。建议读者在虚拟机 Spark03 中运行 Flume Agent 和 Python 程序较长时间，以生成更多的历史用户行为数据，从而使后续的数据分析结果更加丰富。

### 2. 启动采集实时用户行为数据的 Flume Agent

在虚拟机 Spark03 中启动 Flume Agent 的操作步骤如下。

(1) 启动 HDFS 集群、ZooKeeper 集群和 Kafka 集群，确保虚拟机 Spark01、Spark02 和 Spark03 中这些集群的相关进程正常启动。需要说明的是，为了优化资源利用，在采集实时用户行为数据时，可以不启动集群环境中的 YARN 集群、Doris 集群和 Hive 的相关服务。

(2) 参考第 2 章上传 JDK 安装包的方式，将 Python 文件 generate_user_data_real.py 上传到虚拟机 Spark03 的/export/servers 目录中。

(3) 在 Kafka 中创建主题 user_behavior_topic。在虚拟机 Spark01 执行如下命令。

```
kafka-topics.sh --create --topic user_behavior_topic \
--partitions 3 --replication-factor 2 \
--bootstrap-server spark01:9092,spark02:9092,spark03:9092
```

通过上述命令在 Kafka 中创建的主题 user_behavior_topic 包含 3 个分区和 2 个副本，以提升处理效率和数据的容错性。上述命令创建的主题需要与配置文件 flume-logs-

real.conf 中指定的 Kafka 主题一致。

上述命令执行完成后,若出现“Created topic user_behavior_topic”的提示信息,说明在 Kafka 中成功创建主题 user_behavior_topic。

(4) 在 Tabby 中创建一个虚拟机 Spark03 的新操作窗口,用于执行 Python 文件 generate_user_data_real.py,向/export/data/log/2024 目录的日志文件 user_behaviors.log 中写入实时用户行为数据,具体命令如下。

```
python /export/servers/generate_user_data_real.py
```

(5) 在 Tabby 中创建一个虚拟机 Spark03 的新操作窗口,用于启动 Flume Agent,从/export/data/log/2024 目录中的日志文件 user_behaviors.log 里采集实时用户行为数据,具体命令如下。

```
flume-ng agent --name a2 --conf conf/ --conf-file \
/export/data/flume_conf/flume-logs-real.conf \
-Dflume.root.logger=INFO,console
```

上述命令中,参数--name 指定的参数值 a2 为 Flume Agent 的标识,该标识需要与配置文件 flume-logs-real.conf 中 Flume Agent 的标识一致。

在虚拟机 Spark01 中启动一个 Kafka 消费者,该消费者订阅主题 user_behavior_topic,用于验证 Flume 是否将采集的用户行为数据写入 Kafka 的主题 user_behavior_topic 中,具体命令如下。

```
kafka-console-consumer.sh --topic user_behavior_topic \
--group user_behavior_test \
--bootstrap-server spark01:9092,spark02:9092,spark03:9092
```

上述命令执行完成后的效果如图 3-7 所示。

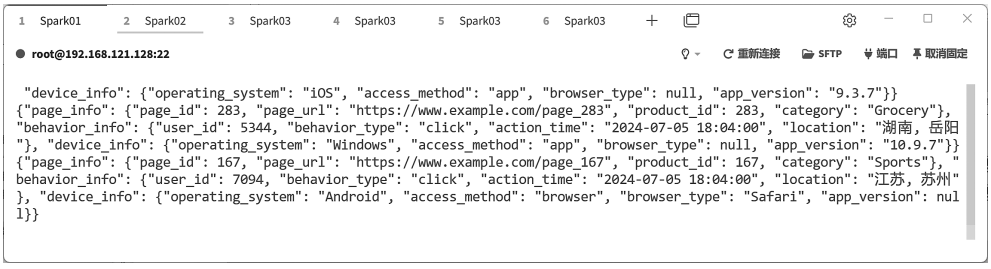


图 3-7 Kafka 消费者

从图 3-7 可以看出,Kafka 消费者输出了生成的用户行为数据,说明 Flume 成功将采集的用户行为数据写入 Kafka 的主题 user_behavior_topic 中。

**小提示:** 在确认 Flume 成功将采集的用户行为数据写入 Kafka 的主题 user_behavior_topic 后,读者可以暂时关闭生成实时用户行为数据的 Python 程序,以及负责采集实时用户行为数据的 Flume Agent。待后续进行实时分析时,再重新启动它们。

在关闭 Flume Agent、Python 程序或者 Kafka 消费者时，可以在相应的操作窗口中，通过组合键 Ctrl ＋ C 实现。

**脚下留心：调整 Flume Agent 可使用 JVM 的最大内存**

当 Flume Agent 启动后，出现 OutOfMemoryError 的错误信息时，通常是由于采集的数据量较大，导致 Flume Agent 可使用 JVM 的内存不够用所导致。读者可以通过修改配置文件 flume-env.sh，调整 Flume Agent 启动和运行时可使用 JVM 的最大内存，具体操作步骤如下。

(1) 通过复制模板文件 flume-env.sh.template 创建配置文件 flume-env.sh。在 Flume 安装目录的/conf 目录中执行如下命令。

```
cp flume-env.sh.template flume-env.sh
```

(2) 使用 vi 编辑器编辑配置文件 flume-env.sh，在文件的末尾添加如下内容。

```
#根据实际情况填写 JDK 安装目录
export JAVA_HOME=/export/servers/jdk1.8.0_401/
export JAVA_OPTS="-Xms1024m -Xmx2048m -Dcom.sun.management.jmxremote"
```

上述内容指定 Flume Agent 启动和运行时可使用 JVM 的最大内存分别为 1GB(-Xms1024m)和 2GB(-Xmx2048m)。配置文件 flume-env.sh 的内容修改完成后，保存并退出编辑。

## 3.5　本章小结

本章主要讲解了数据采集的相关内容。首先，介绍了用户行为数据的概念。然后，介绍了模拟生成用户行为数据。最后，分别介绍了采集方案的配置以及如何采集用户行为数据。通过本章的学习，读者应可以掌握项目中数据采集的实现，为后续实施数据分析提供数据支撑。