第 5 章 高效存储和检索日志记录

CHAPTER 5

随着日志文件大小的增长，单个文本文件可能变得非常庞大，导致读取和写入的性能下降。尤其是在处理大量日志数据时，直接使用文本文件进行存储和检索会变得非常低效。同时，日志文件本身不具备高效查询的能力。每次检索日志时，必须从文件中逐行读取并进行匹配，这对于大文件或大规模日志量的场景来讲查询性能非常差。为了更高效地处理、存储和分析日志数据，采用更专业的数据库系统或日志管理平台是一个更好的选择。接下来，本书将阐述关于使用 MongoDB 数据库存储日志记录和 Elasticsearch 引擎检索日志记录的相关内容。

5.1 基于 MongoDB 存储记录

MongoDB 是一个开源、面向文档的非关系（Not Only SQL，NoSQL）数据库系统，被广泛地用于存储结构化或半结构化数据。当然，它与传统的关系数据库（Relational Database Management System，RDBMS）相比，有许多不同的设计理念，特别是在存储结构、扩展性和查询模型方面。MongoDB 是一个高性能、高可扩展性的数据库，适用于需要快速开发、大规模存储和快速检索的应用场景。

5.1.1 MongoDB 的存储结构

在 MongoDB 数据库中，采用层级结构来存储数据，包括实例（Instance）、数据库（Database）、集合（Collection），以及文档（Document），如图 5-1 所示。

其中，实例是指运行中的 MongoDB 进程，它本身不涉及具体的数据存储，而是提供了数据访问、查询处理、管理功能、数据持久化等服务。在生产环境中，通常会部署多个 MongoDB 实例来实现高可用性和扩展性。当然，一个实例可能包含多个数据库。

数据库是数据存储的逻辑组织单位，即存储数据的容器。虽然 MongoDB 允许在同一个实例上创建多个数据库，但是每个数据库只能存在于一个实例中。不同的应用程序或功能模块可以使用不同的数据库来达到隔离数据的目的。同时，MongoDB 中的数据库名称是唯一的，名称的长度不能超过 64 字符，并且名称不能与已有数据库重名。

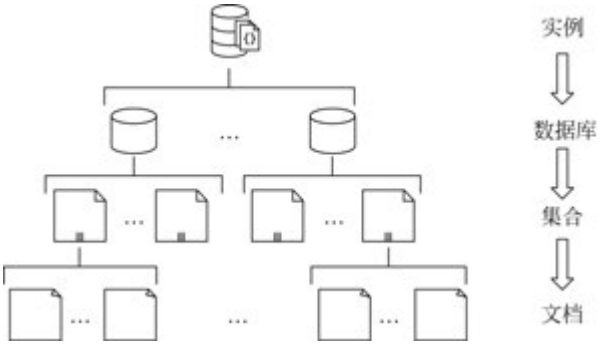

图 5-1　MongoDB 数据库的层级结构

集合是数据库中的数据存储单位，类似于关系数据库中的表，每个数据库可以包含多个集合，并且集合中的数据通常是文档的汇总。同时，MongoDB 不强制要求集合内的文档遵循相同的模式，即每个集合中的文档可能结构不同。

文档是存储的实际数据单位。在一个集合中，文档可以具有完全不同的字段结构，这使 MongoDB 在处理动态变化数据时非常灵活。同时，每个文档都有一个唯一的_id 字段，该字段可以由 MongoDB 自动生成，或者由用户指定，例如，在 users 集合中存储了两个文档，分别包含不同的字段，代码如下：

```
{
  "_id": ObjectId("5f84f8c8a7c7e7f9c8b8b8b8"),
  "name": "Alice",
  "age": 30,
  "email": "alice@example.com"
}

{
  "_id": ObjectId("5f84f8c8a7c7e7f9c8b8b8b9"),
  "name": "Bob",
  "age": 25,
  "phone": " + 1234567890"
}
```

其中，第 1 个文档包含字段_id、name、age、email，而第 2 个文档具有_id、name、age、phone 字段，每个字段都具有不同的含义来保存相应数据。细心的读者会注意到 MongoDB 文档的存储格式类似于 JSON，但它实际上是 BSON 格式，该格式具备更高的性能，并且支持对数据类型进行扩展。当 MongoDB 与应用程序或客户端交互时，它会将 BSON 转换为 JSON 格式，便于阅读。

注意：在 MongoDB 中，_id 字段作为每个文档的唯一标识符，默认由 MongoDB 自动生成，其值通常是 ObjectId 类型，这是 MongoDB 独有的数据类型。

当然，本书仅涉及 MongoDB 数据库的部分内容，更多深入知识读者可自行查阅相关资料。接下来，本书将阐述基于 Docker 容器搭建 MongoDB 数据库环境的相关内容。

5.1.2　搭建 MongoDB 数据库环境

Docker 是一个开源的容器化平台，用于自动化应用程序的部署、扩展和管理。通过 Docker，开发者可以将应用程序及其所有依赖单元打包成一个容器镜像，然后在任何支持 Docker 的平台上运行容器，从而实现应用的一致性、可移植性和高效性。接下来，本书将介绍如何通过 Docker 容器来安装 MongoDB 数据库。

首先，在终端窗口中执行 docker 命令安装并运行 MongoDB 数据库，命令如下：

```
docker run -d --network host --name mongodb mongo:7.0.14
```

其中，docker run 命令用于创建并启动一个新的 Docker 容器。参数-d 能够使容器在后台运行。在默认情况下，docker run 会在前台运行容器，使用参数-d 后，容器会在后台运行，并返回容器的唯一标识 ID。参数--network 用于指定网络模式，包括桥接模式、主机模式等，其中，主机模式由 host 表示，该模式会使容器与主机共享相同的 IP 地址和端口，因此容器中的服务会监听主机的 IP 和端口，没有网络隔离，容器中的应用直接访问主机的网络资源。参数--name 用于为启动的容器指定一个名字，这里将名称指定为 mongodb，通过该名称，用户可以方便地管理该容器，例如，启动、停止、删除容器等操作。参数 mongo 表示将要下载及安装的镜像文件名称。如果不指定 mongo 的版本号，Docker 会自动拉取并使用最新的官方 MongoDB 镜像。如果在终端窗口中成功地执行了上述命令，则会自动拉取版本为 7.0.14 的 MongoDB 镜像并启动容器，该容器将会在后台运行，并且将 MongoDB 默认的 27017 端口映射到本机的 27017 端口，如图 5-2 所示。

```
└─$ sudo docker run -d -p 127.0.0.1:27017:27017 --name mongodb mongo:7.0.14
Unable to find image 'mongo:7.0.14' locally
7.0.14: Pulling from library/mongo
6414378b6477: Pull complete
d5f1c01a7ebb: Pull complete
7c9096921d54: Pull complete
8de1f8e4e0aa: Pull complete
c3c840e83f39: Pull complete
a725a2af1a73: Pull complete
3d555933d128: Pull complete
4e04312eb23d: Pull complete
Digest: sha256:0032d2ca20db5fa34926f196c8a43b74e34ed239a7f2453ff1505b6f12ba8ea6
Status: Downloaded newer image for mongo:7.0.14
ca4406db34b555bedfe78f3f9bd004017bbbca544fdec505df8e3160205a5b6b
```

图 5-2　成功拉取 mongo 镜像，并在后台运行

接下来，通过执行 docker ps 命令来验证是否成功安装并启动 mongo 容器，如图 5-3 所示。

```
└─$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                      NAMES
ca4406db34b5   mongo:7.0.14   "docker-entrypoint.s…"   43 seconds ago   Up 42 seconds   127.0.0.1:27017->27017/tcp   mongodb
```

图 5-3　查看 mongo 镜像运行状态

同时，用户也可以使用 docker 命令切换到容器的命令行中，并查看 MongoDB 数据库的版本信息，从而验证是否成功地安装了 MongoDB 数据库容器，命令如下：

```
docker exec -it mongodb bash
mongod --version
```

如果在 Kali Linux 的终端窗口中成功地执行了上述命令，则会输出 MongoDB 数据库的版本信息，如图 5-4 所示。

图 5-4　查看 MongoDB 服务器的版本信息

最后，使用 MongoDB 客户端可以连接并使用数据库。由于 Kali Linux 默认并未安装该客户端程序，因此需要使用 apt 命令来安装它，命令如下：

```
apt install mongodb-clients
```

如果在终端中成功地执行了 apt 安装命令，则会自动安装 MongoDB 客户端及其依赖软件，如图 5-5 所示。

图 5-5　成功执行 apt 命令安装 MongoDB 客户端及其依赖软件

由于 MongoDB 服务器端与客户端的版本必须兼容才能正常连接，因此笔者通常会确定客户端与 Docker 容器运行的服务器端是否版本一致。此时，通过执行 mongo-version 命令来查看客户端的版本信息，如图 5-6 所示。

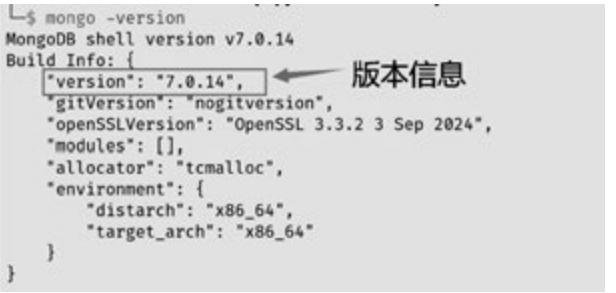

图 5-6　查看 MongoDB 客户端版本信息

细心的读者会发现 MongoDB 客户端与 Docker 容器内运行的服务器端版本一致，因此用户可以使用 mongo 命令相应的客户端来连接 Docker 容器中的 MongoDB 服务器端，命令如下：

```
mongo --host 127.0.0.1 --port 27017
```

如果在终端中成功地连接了 Docker 容器中的 MongoDB 服务器端，则会显示成功连接 MongoDB 的提示信息，如图 5-7 所示。

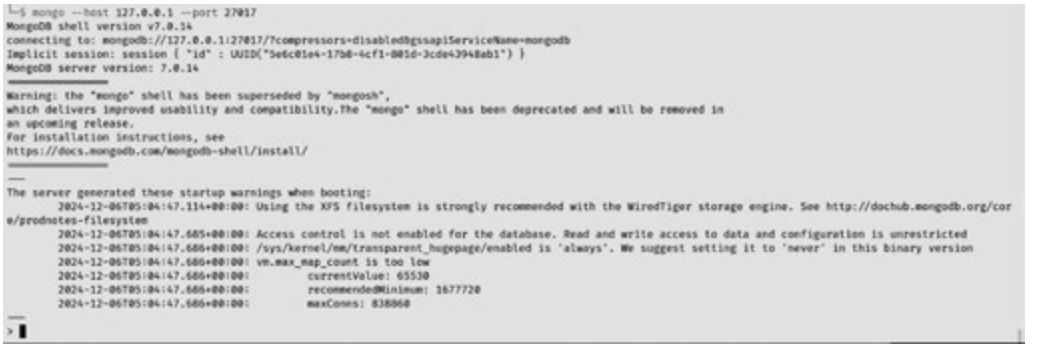

图 5-7　使用客户端成功连接 MongoDB 服务器端

同时，MongoDB 会返回一个能够执行数据库命令的终端，在该终端中，用户可以执行数据库相关指令来管理 MongoDB，例如，执行 show dbs 指令能够查看数据库列表，如图 5-8 所示。

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

图 5-8　查看 MongoDB 数据库列表

虽然用户可以通过终端命令手动管理 MongoDB，但这种方式效率较低，并且难以实现自动化。为了提高管理效率，Python 提供了第三方模块 pymongo，它封装了 MongoDB 操作命令，使自动化管理数据库变得更加简单和高效。接下来，本书将详细介绍如何使用 Python 与 MongoDB 进行交互。

5.1.3 Python 与 MongoDB 交互

作为 Python 与 MongoDB 数据库交互的第三方模块 pymongo，它提供了全面的接口，使 Python 开发者能够轻松地实现对 MongoDB 的各种操作，包括数据库的创建、文档的增、删、改、查、聚合查询等。通过 pymongo 模块，开发者能够高效地与 MongoDB 进行数据交互，简化了连接管理，并支持 MongoDB 的所有核心功能，例如，索引管理、事务支持和数据持久化等。

由于 Python 解释器并未默认集成 pymongo 模块，因此必须安装该模块才能使用其功能。关于在 PyCharm 中安装第三方模块的方法，读者可以查阅本书的 1.1.4 节的相关内容。如果成功地安装了 pymongo 模块，则可以在 PyCharm 内的 Settings 窗口的 Python Interpreter 标签页中查看该模块的相应信息，如图 5-9 所示。

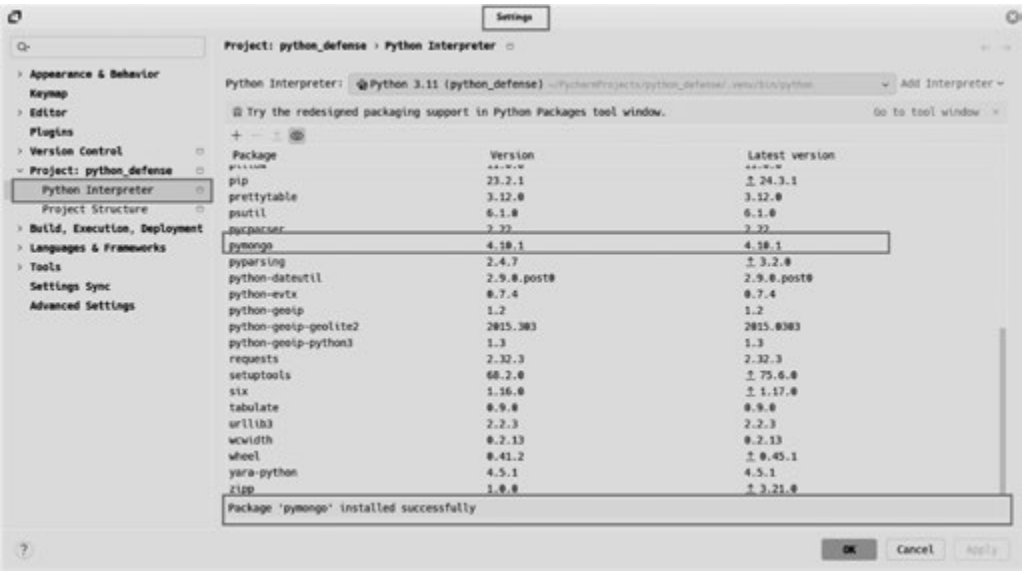

图 5-9 成功安装 pymongo 第三方模块

在 pymongo 模块中，提供了多个函数用于管理 MongoDB 数据库，涵盖了从基础操作到高级功能的各种需求，然而，使用 pymongo 模块进行数据库操作的首要步骤是建立与 MongoDB 实例的连接。用户可以通过初始化 MongoClient 对象来实现连接，代码如下：

```
//ch05/mongodb_connect.py
1  from pymongo import MongoClient
```

```
2  from pymongo.errors import ServerSelectionTimeoutError
3  try:
4      client = MongoClient("mongodb://127.0.0.1:27017/", serverSelectionTimeoutMS =
       5000)
5      client.admin.command('ping')
6      print("MongoDB 连接成功!")
7      client.close()
8  except ServerSelectionTimeoutError:
9      print("MongoDB 连接失败!无法在规定时间内连接到服务器.")
10 except Exception as e:
11     print(f"发生错误: {e}")
```

第 1 行代码表示从 pymongo 模块中导入 MongoClient 类，它用于连接和操作 MongoDB 服务器，即连接 MongoDB 的实例。同时，便于后续代码可以用它来创建 MongoDB 客户端。第 2 行代码表示从 pymongo 模块中导入 ServerSelectionTimeoutError 异常类，该类用于捕获连接超时错误。第 3～11 行代码表示尝试连接本地的 MongoDB 服务器并验证连接是否成功。首先，使用 MongoClient 对象连接 IP 地址为 127.0.0.1 对应的 27017 端口，即 MongoDB 服务，并通过参数 serverSelectionTimeoutMS 将连接超时设置为 5s。接下来，通过执行 ping 命令来确认 MongoDB 服务器是否会响应连接请求。如果连接成功，则输出“MongoDB 连接成功!”提示信息，并关闭连接。如果连接超时，则捕获 ServerSelectionTimeoutError 异常，并输出“MongoDB 连接失败！无法在规定时间内连接到服务器”提示信息。如果发生其他类型的异常，则捕获并输出具体错误信息，从而帮助用户诊断问题。如果在 PyCharm 中成功地执行了 mongodb_connect.py 源代码文件，则将会向终端中输出“MongoDB 连接成功!”提示信息，如图 5-10 所示。

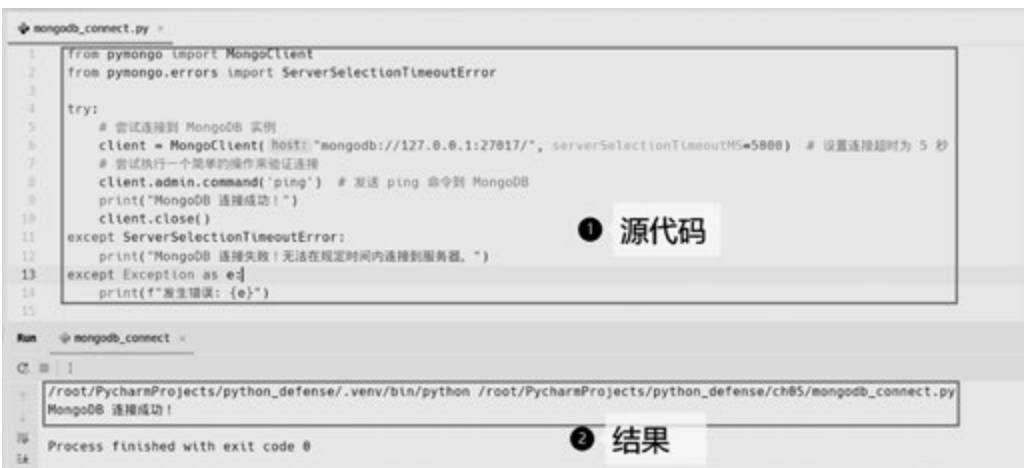

图 5-10　使用 Python 成功连接 MongoDB 服务

当然，用户可以通过建立的连接来向 MongoDB 数据库中插入文档数据，代码如下：

```
//ch05/insert_one_document.py
1  from pymongo import MongoClient

2  client = MongoClient("mongodb://127.0.0.1:27017/")
3  db = client['mydatabase']
4  collection = db['mycollection']
5  data = {"name": "Alice", "age": 30, "city": "New York"}
6  collection.insert_one(data)
7  client.close()
8  print("数据插入成功!")
```

第 1 行代码表示从 pymongo 库导入 MongoClient 类，该类能够创建与 MongoDB 数据库的连接，从而可以执行各种 MongoDB 操作，例如，插入、查询、更新等。第 2 行代码表示创建一个 MongoClient 实例，并连接到 MongoDB 服务器。第 3 行代码表示选择一个名为 mydatabase 的数据库。如果该数据库不存在，则 MongoDB 会在插入数据时自动创建它，其中，db 变量可以引用 mydatabase 数据库。第 4 行代码表示选择一个名为 mycollection 的集合，集合类似于 SQL 中的表。如果该集合不存在，则 MongoDB 会在插入数据时自动创建它，其中，collection 变量可以引用的是 mycollection 集合。第 5 行代码表示定义一个名为 data 的字典变量，它包含要插入 mycollection 集合中的文档。该文档有 3 个字段 name、age 和 city，它们分别代表名字、年龄和城市。第 6 行代码表示调用 MongoClient 对象的 insert_one 方法将 data 文档插入 mycollection 集合中。如果成功插入，则 MongoDB 会自动为文档生成一个唯一的_id 字段。第 7 行代码表示调用 MongoClient 对象的 close 方法，它会关闭 Python 与 MongoDB 数据库的连接。虽然在脚本结束时连接会自动关闭，但显式调用 close 方法是一种良好的做法，它帮助及时释放与数据库连接相关的资源。第 8 行代码表示调用 print 函数来输出一条成功消息，表示数据已成功插入 MongoDB 数据库中。如果在 PyCharm 中成功地执行了 insert_one_document.py 源代码文件，则会向终端输出“数据插入成功!”提示信息，如图 5-11 所示。

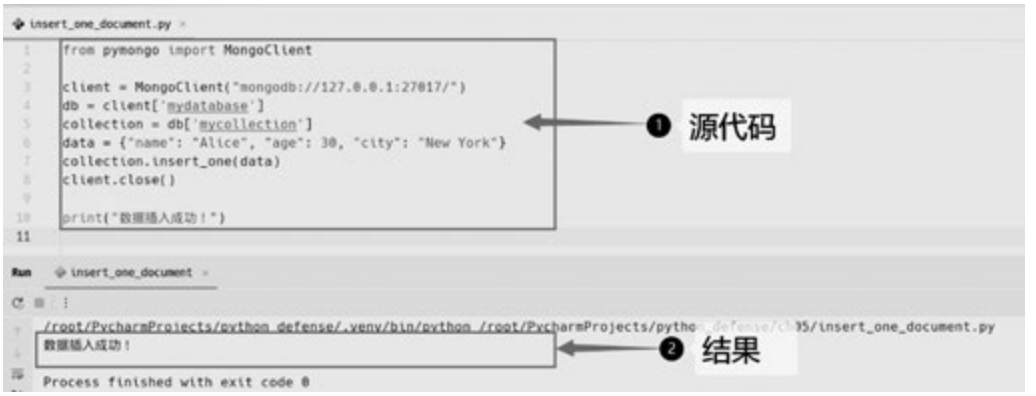

图 5-11　成功执行 insert_one_document.py 源代码文件

同时，用户可以使用 Python 与 MongoDB 的连接来查询文档数据，代码如下：

```
//ch05/find_one_document.py
 1  from pymongo import MongoClient

 2  client = MongoClient("mongodb://127.0.0.1:27017/")
 3  db = client['mydatabase']
 4  collection = db['mycollection']

 5  result = collection.find_one({"name": "Alice"})

 6  if result:
 7      print("查找到的文档: ", result)
 8  else:
 9      print("没有找到匹配的文档.")

10  client.close()
```

第 1～4 行代码表示导入 pymongo 模块中的 MongoClient，通过创建 MongoClient 类的对象来建立与 mongodb 服务的连接，并依次选择相应的数据库和集合。最终，使用变量 collection 可以引用对应的集合。第 5 行代码表示调用 MongoClient 对象的 find_one 方法来查找第 1 个符合查询条件的文档，其中，查询条件是{"name": "Alice"}，即查找 name 字段为 Alice 的文档。如果存在符合查询条件的文档，则返回匹配条件的第 1 条文档，否则返回 None。第 6～9 行代码表示通过判断执行 find_one 函数返回的结果 result 是否存在来决定输出相关提示信息。如果 result 不为空，则表示找到了匹配的文档，并输出该文档的内容，否则说明没有找到符合条件的文档，同时会向终端输出“没有找到匹配的文档”提示信息。如果在 PyCharm 中成功地执行了 find_one_document.py 源代码文件，则会向终端输出“查找到的文档”提示信息，以及该文档对应的内容，如图 5-12 所示。

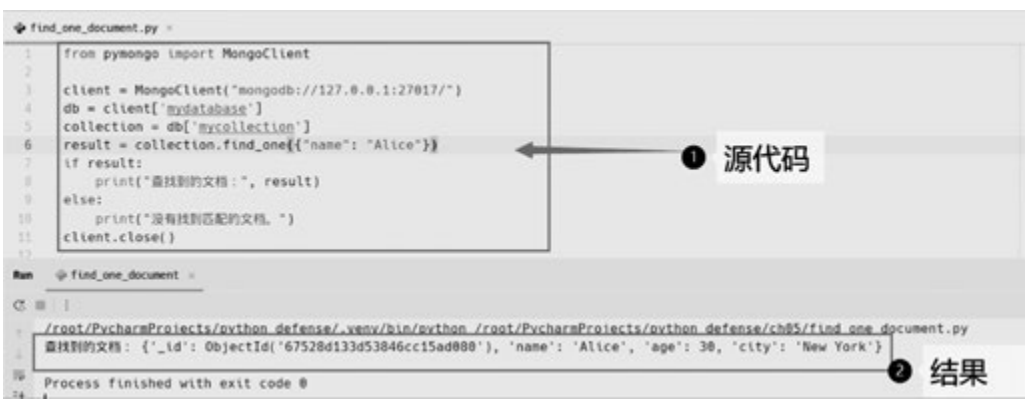

图 5-12　成功查询到符合条件的文档

当然，pymongo 模块还提供了许多其他用于操作 MongoDB 的方法，感兴趣的读者可以通过查阅相关资料进一步地深入学习和掌握。接下来，本书将以分析 smb.log 日志文件为

例，讲解如何将日志记录存储到 MongoDB 中。

5.1.4 存储勒索病毒相关日志记录

勒索病毒(Ransomware)是一类恶意软件，通过限制或完全阻止受害者访问其个人数据来进行攻击，迫使受害者支付赎金以恢复访问权限。较为简单的勒索病毒可能仅通过锁定系统来干扰操作，但更为复杂的版本采用了“加密病毒勒索”(Cryptoviral Extortion)技术，直接加密受害者的文件，致使数据变得无法访问。攻击者要求支付赎金以换取解密密钥，恢复对文件的访问。在这种加密勒索攻击中，一旦文件被加密，若没有解密密钥，恢复数据几乎是不可能的，而使用诸如 Paysafecard、比特币等难以追踪的数字货币作为赎金，使追踪和起诉罪犯变得更加困难。

勒索病毒会加密受害者的文件，并将文件名修改为特定的扩展名，例如，常见的扩展名包括.encrypted、.locked、.wncry 等。通过分析 smb.log 日志文件中是否存在这些扩展名，可以有效地识别勒索病毒的活动。接下来，用户可以根据这一特性能够实现检索 smb.log 日志文件中关于勒索病毒的日志记录，代码如下：

```
//ch05/detect_ransomware.py
 1  import re
 2  from datetime import datetime as dt
 3  from prettytable import PrettyTable


 4  def openlogfile(path):
 5      with open(path) as log_file:
 6          for log_record in log_file:
 7              yield log_record


 8  def parsesmb(log_record):
 9      pattern = r"^(?P<ts>[0-9]{2}:[0-9]{2}:[0-9]{2})\s:\s(?P<client_hostname>
        [a-zA-Z0-9\-]+)\|(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.
        [0-9]{1,3})\|(?P<share>[a-zA-Z0-9\-]+)\|(?P<operation>[a-zA-Z]+)\|
        ok\|(?P<path>.*)$"
10      log_data = re.search(pattern, log_record)
11      if log_data:
12          r = log_data.groupdict()
13          r['ts'] = dt.strptime(r['ts'], "%H:%M:%S")
14          if r['operation'] == 'rename':
15              r['path'] = r['path'].split("|")[-1]
16          return r
17      return None


18  def ransomwareAlert(path="logs/smb.log"):
```

```
    ext_re = r"\.encrypted|\.locked|\.wncry"
    table = PrettyTable()
    table.field_names = ["Timestamp", "Client Hostname", "Client IP", "Share", "Operation",
    "Path"]
    smb_log = openlogfile(path)
    for log_record in smb_log:
       try:
          log_data = parsesmb(log_record)
          if log_data and re.search(ext_re, log_data['path']):
             table.add_row([log_data['ts'].strftime('%H:%M:%S'), log_data['client_
             hostname'], log_data['client_ip'],
                            log_data['share'], log_data['operation'], log_data['path']])
       except Exception as e:
        pass
    print(table)


ransomwareAlert()
```

第 1 行代码表示导入 re 模块，它用于匹配正则表达式来进行搜索文本。第 2 行代码表示从 datetime 模块导入 datetime 类并将其重命名为 dt，它用于处理日期和时间。第 3 行代码表示导入 prettytable 模块，它用于生成美化格式的表格，便于显示日志内容。第 4～7 行代码表示定义一个名为 openlogfile 的函数，它用于打开指定路径的日志文件，并逐行读取文件内容。最终，通过 yield 关键字返回每行日志记录，使该函数成为一个生成器，能够按需逐条读取文件中的内容，而不是一次性将整个文件加载到内存中。第 8～17 行代码表示定义一个名为 parsesmb 的函数，它能够解析 smb.log 日志文件的一条记录，并将记录中的不同字段和相应值添加到字典变量 r 中，同时将其作为函数的返回值。第 18 行代码表示定义一个名为 ransomwareAlert 的函数，它用于分析指定的 SMB 日志文件，识别其中可能与勒索病毒相关的记录，并将符合条件的日志信息以表格形式输出。同时，这个函数接受一个可选参数 path，其默认值为 logs/smb.log，它表示日志文件的路径。第 19 行代码表示定义一个保存正则表达式的变量 ext_re，它用于匹配常见的勒索病毒文件扩展名。第 20 行代码表示创建一个名为 table 的 PrettyTable 对象，它用于格式化输出表格数据。第 21 行代码表示设置表格的列名称，用于显示日志中的不同字段，其中，列名称包括时间戳、客户端主机名、客户端 IP 地址、共享名称、操作类型和路径。第 22 行代码表示调用 openlogfile 函数读取指定路径的日志文件，并返回一个生成器，该生成器按行读取日志文件。同时，使用变量 smb_log 保存生成器。第 23 行代码表示遍历 smb_log 生成器中的每行日志记录，每次迭代获取一条日志记录，并将其保存到变量 log_record 中。第 24 行代码表示开始一个 try 代码块，以便捕捉并处理任何可能的异常。第 25 行代码表示调用 parsesmb 函数来解析当前日志记录 log_record，并将解析后的数据保存到字典变量 log_data 中。第 26 行代码表示检查 log_data 是否为空，并且 log_data 中 path 字段对应的路径值是否匹配正则表达式 ext_re。

如果成功匹配，则继续执行后续代码，否则跳过后续代码并循环解析下一条记录。第 27 行代码表示将日志记录中的数据以表格的形式添加到名为 table 的 PrettyTable 对象中。第 28 行和第 29 行代码表示捕捉可能发生的任何异常，如果捕获到异常，则执行 pass 语句，即跳过当前日志记录并继续处理下一条记录。第 30 行代码表示在所有日志处理完成后，打印表格。如果在 PyCharm 中成功地执行了 detect_ransomware.py 源代码文件，则会向终端输出符合勒索病毒特征的日志记录表格，如图 5-13 所示。

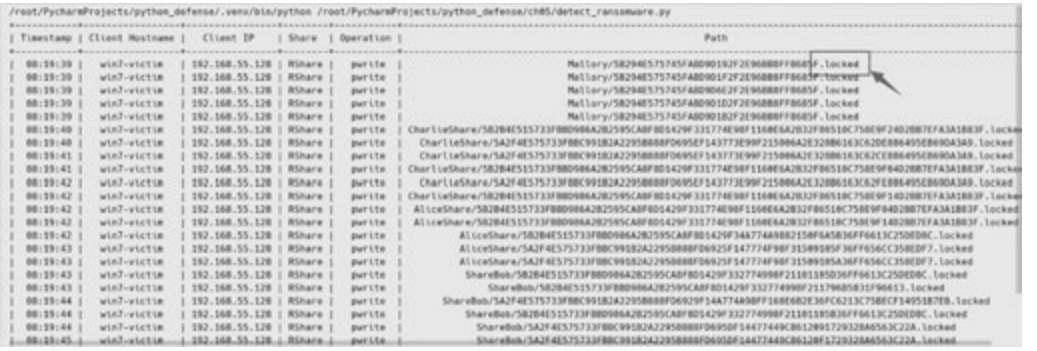

图 5-13　输出符合勒索病毒特征的日志记录表格

Path 字段中包含了扩展名为 .locked 的文件，这表明这些文件已经被勒索病毒加密。对于文件大小适中的 smb.log 日志文件，用户可以直接通过文件操作进行分析，然而，随着 smb.log 文件不断增大，直接分析变得更加困难。在这种情况下，将特定日志记录存储到 MongoDB 中可以更高效地进行数据管理和分析，代码如下：

```
//ch05/detect_ransomware_2.py
 1  import re
 2  from datetime import datetime as dt
 3  from pymongo import MongoClient
 4  from prettytable import PrettyTable



 5  def openlogfile(path):
 6      with open(path) as log_file:
 7          for log_record in log_file:
 8              yield log_record


 9  def parsesmb(log_record):
10      pattern = r"^(?P<ts>[0-9]{2}:[0-9]{2}:[0-9]{2})\s:\s(?P<client_hostname>[a
        -zA-Z0-9\-]+)\|(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
        9]{1,3})\|(?P<share>[a-zA-Z0-9\-]+)\|(?P<operation>[a-zA-Z]+)\|ok\|(?
        P<path>.*)$"
11      log_data = re.search(pattern, log_record)
```

```
    if log_data:
        r = log_data.groupdict()
        r['ts'] = dt.strptime(r['ts'], "%H:%M:%S")
        if r['operation'] == 'rename':
            r['path'] = r['path'].split("|")[-1]
        return r
    return None


def ransomwareAlert(path="logs/smb.log"):
    ext_re = r"\.encrypted|\.locked|\.wncry"
    table = PrettyTable()
    table.field_names = ["Timestamp", "Client Hostname", "Client IP", "Share", "Operation",
    "Path"]
    client = MongoClient("mongodb://localhost")
    db = client['alerts']
    col = db['smb']
    encrypted_files_count = 0
    smb_log = openlogfile(path)
    for log_entry in smb_log:
        try:
            log_data = parsesmb(log_entry)
            if log_data and re.search(ext_re, log_data['path']):
                col.insert_one(log_data)
                encrypted_files_count += 1
                table.add_row([log_data['ts'].strftime('%H:%M:%S'), log_data['client_
                hostname'], log_data['client_ip'],
                            log_data['share'], log_data['operation'], log_data['path']])
        except Exception as e:
            pass
    print(f"\nNumber of encrypted files: {encrypted_files_count}")
    print(f"\nNumber of encrypted files: {col.count_documents({})}")
    print(table)

ransomwareAlert()
```

第 23 行代码表示创建一个名为 client 的 MongoDB 客户端对象，并连接到运行在本地的 MongoDB 实例。第 24 行代码表示通过 client 对象访问名为 alerts 的数据库。如果该数据库不存在，则 MongoDB 会自动创建它。第 25 行代码表示选择名为 smb 的集合，集合是存储文档的地方，相当于关系数据库中的表。如果该集合不存在，则 MongoDB 会自动创建它。第 32 行代码表示调用 insert_one 方法将日志记录插入 MongoDB 的文档中来保存该条记录。第 33 行代码表示使用变量 encrypted_files_count 来计算勒索软件加密的文件个数。第 37 行代码表示输出变量 encrypted_files_count 中保存的文件个数。第 38 行代码表示调用 print 函数输出由 count_documents 统计的文档个数。

如果在 PyCharm 中成功地执行了 detect_ransomware_2.py 源代码文件，则会输出勒

索软件加密的文件总数，以及符合勒索病毒特征的日志记录表格，如图 5-14 所示。

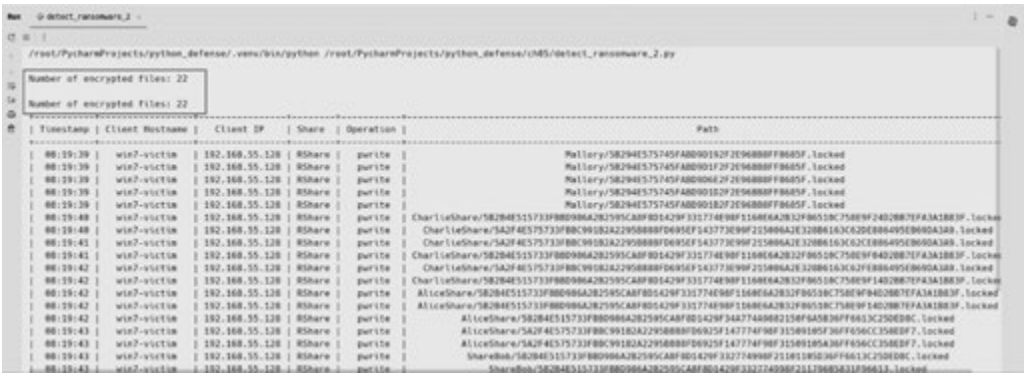

图 5-14　成功执行 detect_ransomware_2. py 源代码文件

在日志分析过程中，虽然 MongoDB 提供了强大的存储和检索功能，但其并不具备直接的可视化功能，这对于需要直观显示复杂数据的场景存在一定的局限性。为了弥补这一不足，接下来，本书将阐述基于 Elasticsearch 检索日志记录，并利用 Kibana 对结果进行直观显示的相关内容。

5.2　基于 Elasticsearch 检索记录

Elasticsearch 是一个分布式搜索和分析引擎，专门用于存储、索引和查询日志数据，能够高效地处理大规模数据并提供强大的全文搜索和分析功能，而 Kibana 是一个基于 Web 的数据可视化平台，使用户能够直观地查看和分析存储在 Elasticsearch 中的日志数据，从而帮助开发人员和运维人员监控和调试系统。

5.2.1　搭建 Elasticsearch 和 Kibana 环境

虽然用户可以通过 Docker 命令逐条拉取 Elasticsearch 和 Kibana 镜像并启动相应容器，但这种逐条执行命令的方式不仅烦琐，还容易因输入错误而导致环境搭建失败，因此笔者通常采用 Docker Compose 工具，通过配置该工具实现一键启动多个容器，从而简化操作并降低出错的风险。

Docker Compose 是一个用于定义和管理多容器 Docker 的工具。在使用 Docker Compose 工具时，用户可以通过配置文件来设定应用的服务、网络、数据卷等，并使用命令一键启动所有服务。在使用 Compose 工具时，通常包括两个核心步骤，即定义和启动服务。

首先，定义服务是指通过 docker-compose. yml 文件来定义多个服务及其属性，包括每个服务的 Docker 镜像、环境变量、端口映射、依赖关系等，例如，配置 Elasticsearch 和 Kibana 容器的 docker-compose. yml 文件，代码如下：

```
//ch05/docker-compose.yml
version: '3.7'

services:
  #Elasticsearch Docker Images: https://www.docker.elastic.co/
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.3.3
    container_name: elasticsearch
    environment:
      - xpack.security.enabled=false
      - discovery.type=single-node
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    cap_add:
      - IPC_LOCK
    volumes:
      - elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
      - 9300:9300

  kibana:
    container_name: kibana
    image: docker.elastic.co/kibana/kibana:8.3.3
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    ports:
      - 5601:5601
    depends_on:
      - elasticsearch

volumes:
  elasticsearch-data:
    driver: local
```

接下来，启动服务是指根据 docker-compose.yml 配置文件内容来自动拉取镜像的，然后创建并启动相应服务，命令如下：

```
docker-compose up -d
```

命令 docker-compose 是指使用 Docker Compose 工具，管理定义在 docker-compose.yml 配置文件中的服务。参数 up 表示启动并运行 docker-compose.yml 文件中定义的所有服务。

如果容器没有被创建，则它会先构建并启动容器。参数-d 表示分离模式，即容器在后台运行，不会占用当前的终端。在使用该参数选项后，命令执行完毕后终端会立即返回，容器会在后台运行。如果在终端窗口中成功地执行了 docker-compose 命令，则会根据 docker-compose.yml 文件内容自动安装并启动相应容器，如图 5-15 所示。

```
└─$ sudo docker-compose up -d
Creating network "docker_test_default" with the default driver
Creating volume "docker_test_elasticsearch-data" with local driver
Pulling elasticsearch (docker.elastic.co/elasticsearch/elasticsearch:8.3.3)...
8.3.3: Pulling from elasticsearch/elasticsearch
5486d10d7ee8: Pull complete
9def63f3a19b: Pull complete
d4b26a2ddc81: Pull complete
22be68412582: Pull complete
4aafc7bb7a4c: Pull complete
e2d2ad8860a3: Pull complete
6a013dc5a317: Pull complete
985f76164874: Pull complete
ce10c698a14d: Pull complete
Digest: sha256:caef7887384d9c77f309508ce72722bf21c7991d5fe81f23eaf843d1ca891fe4
Status: Downloaded newer image for docker.elastic.co/elasticsearch/elasticsearch:8.3.3
Pulling kibana (docker.elastic.co/kibana/kibana:8.3.3)...
8.3.3: Pulling from kibana/kibana
5486d10d7ee8: Already exists
3bc723d98318: Pull complete
986f201d2fa0: Pull complete
3fa890c75602: Pull complete
e3ff8f4c0004: Pull complete
89732bc75041: Pull complete
3dffa41694bb: Pull complete
88a6e4779f78: Pull complete
10b6a097ddf2: Pull complete
d6825a69ecce: Pull complete
f57b0d59315c: Pull complete
1d65d81c1759: Pull complete
e1c66daacb58: Pull complete
0d48eb2aab55: Pull complete
Digest: sha256:51d6233517d2e4cabf2bd180ced55293103325ee9db690b68732e3cc272614e0c5
Status: Downloaded newer image for docker.elastic.co/kibana/kibana:8.3.3
Creating elasticsearch ... done
Creating kibana        ... done
```

图 5-15　自动安装并启动 Elasticsearch 和 Kibana 容器

如果成功地启动了 Kibana 容器，则可以使用浏览器访问 127.0.0.1:5601 链接地址来打开该服务提供的 Web 页面，如图 5-16 所示。

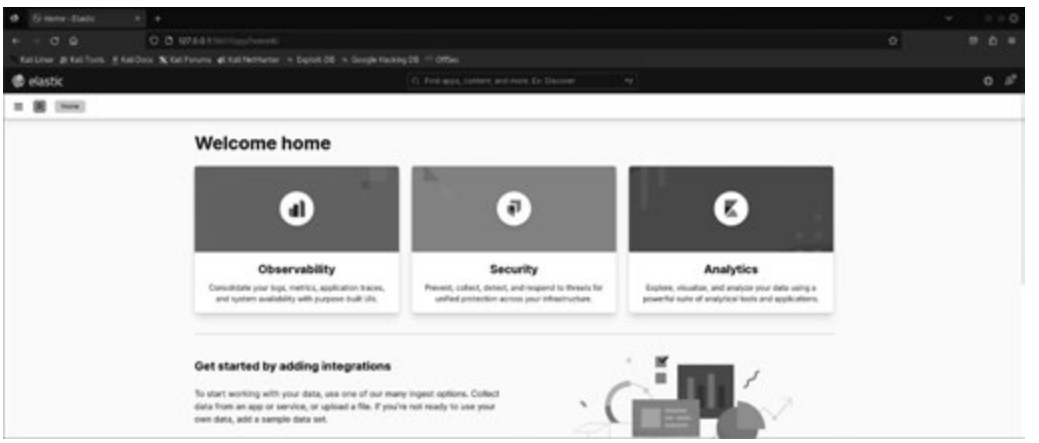

图 5-16　Kibana 服务页面

由于 Elasticsearch 本身不提供 Web 界面，因此用户无法直接通过浏览器访问该服务，然而，Kibana 是与 Elasticsearch 配合使用的 Web 界面工具，它提供了一个基于 Web 的前端，允许用户通过浏览器可视化和分析存储在 Elasticsearch 中的数据，其中，Kibana 提供的仪

表盘、图表和查询功能能够帮助用户更直观地展示和分析 Elasticsearch 数据，如图 5-17 所示。

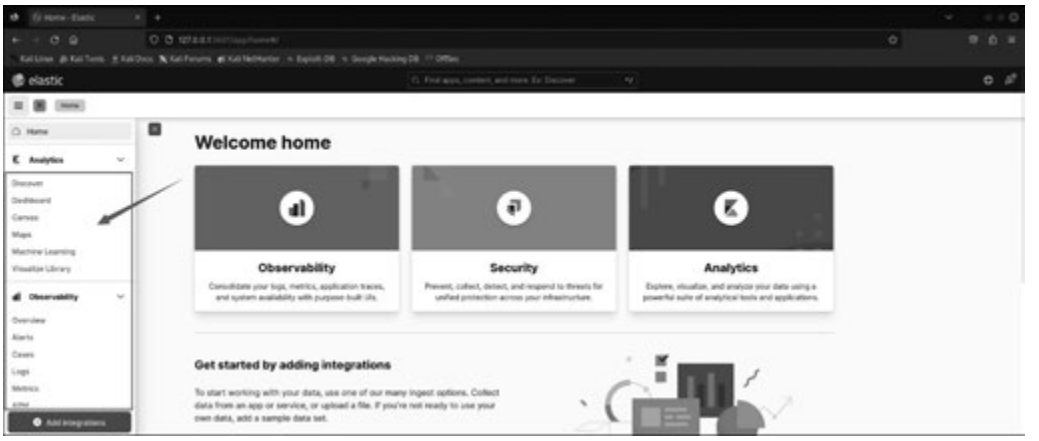

图 5-17　Kibana 提供的仪表盘、图表等功能面板

虽然通过浏览器可以直接访问 Kibana 服务，但由于 Kibana 连接的 Elasticsearch 服务尚无数据，无法充分发挥其日志分析功能，因此需要先将数据导入 Elasticsearch 中，以便 Kibana 能够有效地分析和展示日志数据。接下来，本书将介绍使用 Python 与 Elasticsearch 模块进行交互的相关内容。

5.2.2　Python 与 Elasticsearch 交互

Python 语言中的 Elasticsearch 第三方模块是一个 Elasticsearch 官方提供的客户端库，它主要实现与 Elasticsearch 引擎进行交互的功能。通过该模块，用户可以高效地执行数据索引、查询、更新和管理等操作，适用于日志分析、全文搜索、数据分析等多个场景。

由于 Python 语言默认并未集成 Elasticsearch 模块，因此使用的前提是成功安装该模块。在 PyCharm 的 Settings 窗口中，通过 Python Interpreter 标签页可以安装 Elasticsearch 模块。如果成功地安装了这个模块，则会输出已安装模块的提示信息，如图 5-18 所示。

接下来，用户可以使用 Elasticsearch 模块来连接 Elasticsearch 服务的 9200 端口并输出该服务的信息，从而验证是否成功启动 Elasticsearch 服务，代码如下：

```
//ch05/connect_Elasticsearch.py
1  from elasticsearch import Elasticsearch
2  es = Elasticsearch(["http://localhost:9200"])
3  print(es.info())
```

第 1 行代码表示导入 elasticsearch 模块的 Elasticsearch 类，便于后续代码直接使用这个类。第 2 行代码表示创建一个 Elasticsearch 对象，并传入一个列表["http://localhost:9200"]作为参数。这个列表中包含 Elasticsearch 服务的地址和端口，即 localhost 和 9200。变量 es 用于保存一个连接到 Elasticsearch 服务的对象，通过该对象用户可以完成 Elasticsearch

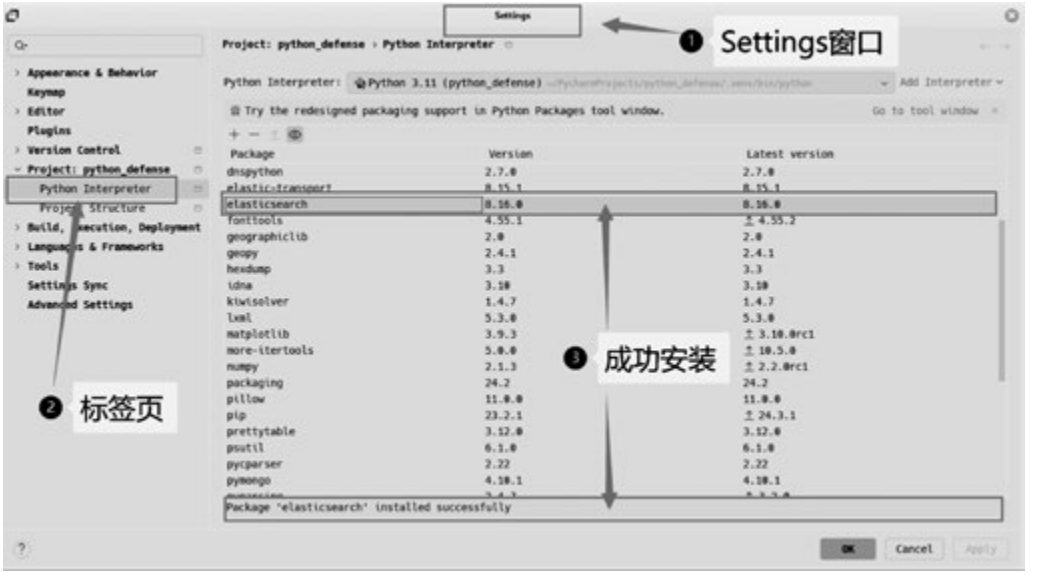

图 5-18　成功安装 Elasticsearch 模块

服务的相关操作。第 3 行代码表示调用 print 和 info 函数向终端输出 Elasticsearch 服务的基本信息，其中，info 方法用于获取 Elasticsearch 服务的基本信息，例如，版本、集群名称等。

如果在 PyCharm 中成功地执行了 connect_Elasticsearch.py 源代码文件，则会向终端窗口输出 Elasticsearch 服务相关信息，即成功连接 Elasticsearch 服务，如图 5-19 所示。

图 5-19　使用 Python 成功连接 Elasticsearch 服务

在 Elasticsearch 中，数据是基于 JSON 格式的文档进行存储和管理的。虽然用户可以使用 Python 语言实现向 Elasticsearch 服务执行增、删、改、查等操作，但是笔者使用 Elasticsearch 模块最多的功能是新建文档，代码如下：

```
//ch05/add_Elasticsearch.py
1  from elasticsearch import Elasticsearch
2  es = Elasticsearch(["http://localhost:9200"])
3  es.index(index = "my_index", id = 1, body = {"name": "Alice", "age": 30})
```

第 3 行代码表示新建一个名为 my_index 的索引，以及名为 1 的 ID，即文档的标识 ID 为 1。同时，将这个文档的值设定为{"name"：Alice", "age"：30}。

注意：索引是 Elasticsearch 中存储和组织文档的逻辑单元，它类似于关系数据库中的表，用于存储和管理文档。每个索引包含一组文档，并且每个索引都有一个名字，用于在查询和操作时进行区分。文档 ID 是每个文档的唯一标识符，它用于区分不同的文档。如果用户没有提供 ID，Elasticsearch 则会自动生成一个唯一的 ID。

如果在 PyCharm 中成功地执行了 add_Elasticsearch.py 源代码文件，则会在 Elasticsearch 服务中新建一个新的文档。同样地，通过 Elasticsearch 模块的 GET 方法，用户可以根据索引和 ID 值获取特定文档，代码如下：

```
//ch05/get_Elasticsearch.py
1  from elasticsearch import Elasticsearch
2  es = Elasticsearch(["http://localhost:9200"])
3  doc = es.get(index = "my_index", id = 1)
4  print(doc['_source'])
```

如果在 PyCharm 中成功地执行了 get_Elasticsearch.py 源代码文件，则会输出该文档的信息，如图 5-20 所示。

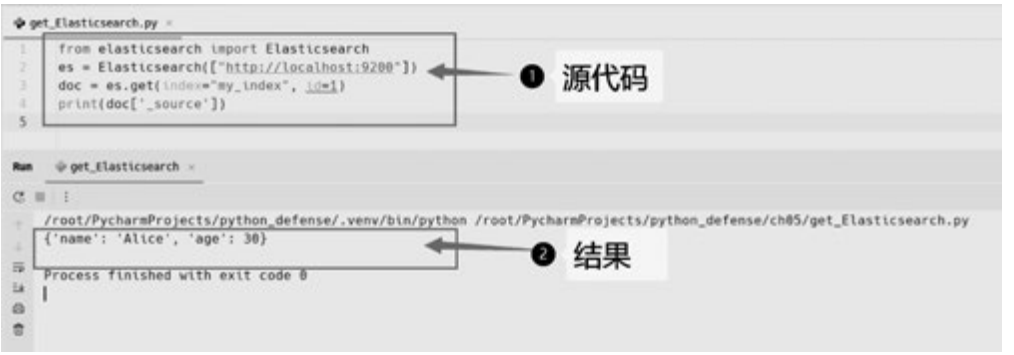

图 5-20　成功读取 Elasticsearch 服务的文档

当然，用户也可以使用浏览器访问 Kibana 服务来查看文档内容，如图 5-21 所示。

图 5-21　查看文档内容

感兴趣的读者也可以自行学习并掌握 Elasticsearch 模块提供的其他方法。接下来，本书将介绍组合 Elasticsearch 和 Kibana 实时分析日志记录的相关内容。

5.2.3 检索勒索病毒相关日志记录

首先，使用 Python 将日志文件 smb.log 的记录导入 Elasticsearch 中，代码如下：

```
//ch05/import_smb.py
import re
from datetime import datetime as dt
from elasticsearch import Elasticsearch
def openlogfile(path):
    with open(path) as log_file:
        for log_record in log_file:
            yield log_record
def parseSmb(log_record):
    pattern = r"^(?P<ts>[0-9]{2}:[0-9]{2}:[0-9]{2})\s:\s(?P<client_hostname>
    [a-zA-Z0-9\-]+)\|(?P<client_ip>[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.
    [0-9]{1,3})\|(?P<share>[a-zA-Z0-9\-]+)\|(?P<operation>[a-zA-Z]+)\|
    ok\|(?P<path>.*)$"
    log_data = re.search(pattern, log_record)
    if log_data:
       r = log_data.groupdict()
       r['ts'] = dt.strptime(r['ts'], "%H:%M:%S")
       if r['operation'] == 'rename':
          r['path'] = r['path'].split("|")[-1]
       return r
    return None
def exportSmbActivity(path="logs/smb.log"):
    es = Elasticsearch("http://localhost:9200")
    log_file = openlogfile(path)
    for log_record in log_file:
       try:
          log_data = parseSmb(log_record)
          if log_data:
             es.index(index="smb", document=log_data)
       except Exception as e:
          print(f"Error processing log record: {e}")
          pass
if __name__ == "__main__":
    exportSmbActivity("logs/smb.log")
```

如果在 PyCharm 中成功地执行了 import_smb.py 源代码文件，则会将 smb.log 日志文件中的记录导入 Elasticsearch 中。接下来，用户可以使用浏览器访问 Kinaba 来验证是否成功导入日志记录。使用 Kibana 分析数据的第 1 步是打开 Discover 工具，这是一个用于浏览和分析 Elasticsearch 中存储数据的工具，并且它能够快速查看、搜索和过滤日志或其他数据，以便进行实时分析。在 Kinaba 页面中，依次选择侧边栏→Discovery 按钮，即可打开

Discover 分析页面，如图 5-22 所示。

图 5-22　打开 Discover 分析页面

接下来，在 Discover 分析页面中，单击 Create data view 按钮可以打开创建索引视图的页面，如图 5-23 所示。

图 5-23　创建索引视图的页面

细心的读者会注意到在索引视图页面中会自动加载 Elasticsearch 中的索引，例如，使用 Python 代码导入的 smb 索引。当然，用户可以为该索引创建视图，通过在 Name 输入框中填写正则表达式来将索引匹配到视图中。同时，用户也能够在 Timestamp field 下拉列表中选择用于标识时间的字段。如果 Name 中的正则表达式成功地匹配到相应索引，则会输出 Your index pattern matches 1 source 的提示信息，如图 5-24 所示。

图 5-24　成功匹配 smb 索引

接下来，通过单击 Create data view 按钮即可完成创建索引任务，并且会自动跳转到 Discover 工具的分析页面。由于 Discover 工具默认设置只显示最近 15min 内的活动，因此会显示 No results match your search criteria 提示信息，如图 5-25 所示。

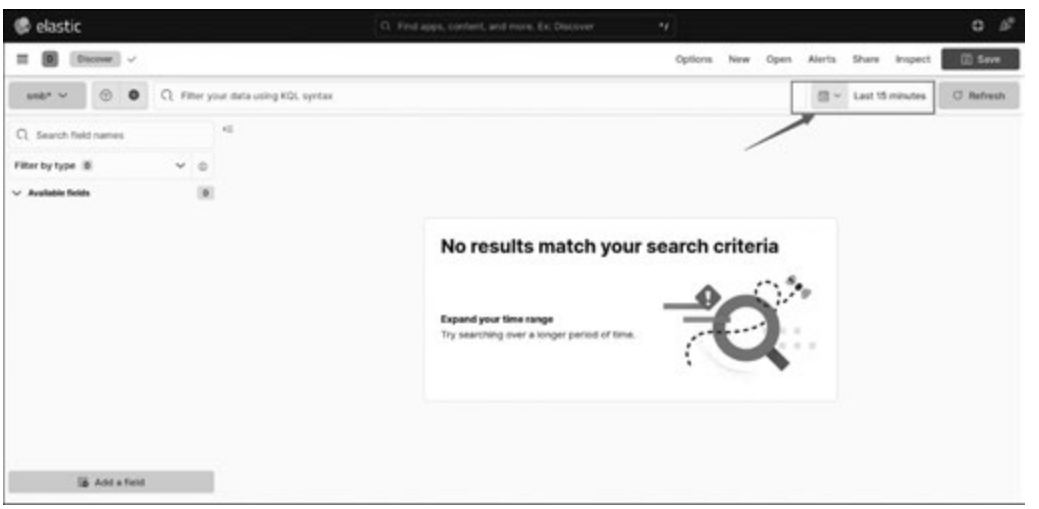

图 5-25　跳转到 Discover 工具的分析页面

通过单击 Quick select 按钮可以打开设置筛选时间的窗口，并选择合适的时间范围来显示相应数据，例如，选择 Today 选项，如图 5-26 所示。

通过单击 Today 按钮，即可将时间范围设定为今天，并在 Discovery 工具的页面中显示所有相关的日志记录，如图 5-27 所示。

最后，通过在 Filter your data using KQL syntax 输入框中填写 path.keyword ： *.locked 作为查询条件，即可筛选所有包含.locked 的日志记录，如图 5-28 所示。

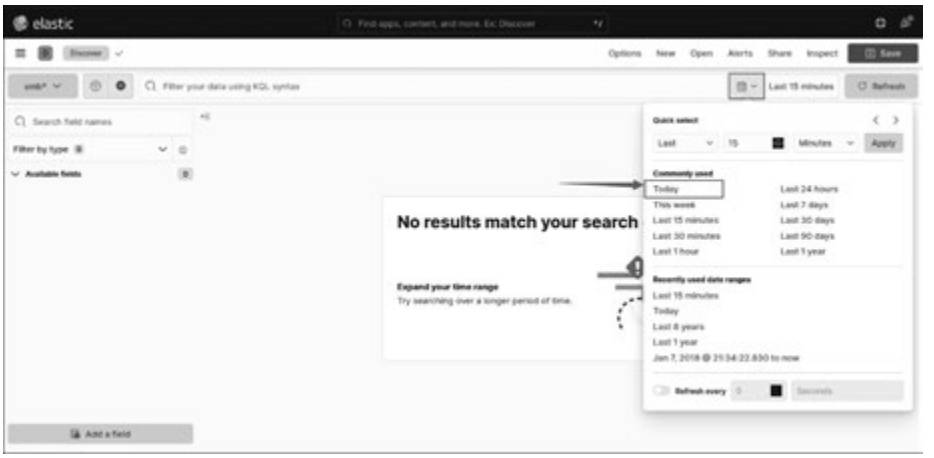

图 5-26　打开选择筛选时间的窗口

图 5-27　显示今天所有的日志记录

图 5-28　成功筛选包含 .locked 的日志记录

同样地，读者还可以基于多种条件筛选日志记录，例如，将筛选条件设置为 path.keyword：*.locked or path.keyword：*.encrypted，即可筛选 path 中所有包含 locked 和 encrypted 的相关记录。显然，通过 Kibana 和 Elasticsearch 工具，用户能够高效地分析日志数据，从而快速地识别潜在问题。接下来，本书将阐述关于检索系统登录日志记录的相关内容。

5.2.4 检索系统登录相关日志信息

首先，使用 Python 将日志文件 auth.log 的记录导入 Elasticsearch 中，代码如下：

```
//ch05/import_auth.py
 1  import re
 2  from datetime import datetime as dt
 3  from elasticsearch import Elasticsearch
 4  from geoip import geoip2
 5  def openlogfile(path):
 6      with open(path) as log_file:
 7          for log_record in log_file:
 8              yield log_record
 9  def parseAuth(log_record):
10      log_data = re.search(
11          r"^(?P<ts>\w{3}\s\d{1,2}\s\d{1,2}:\d{1,2}:\d{1,2})" +
12          r"\s(?P<host>[\w\-]+)" +
13          r"\s(sshd\[\d{1,6}\]):" +
14          r"\s(?P<action>Failed|Accepted) password for(\s(?P<invalid>invalid
            user))?" +
15          r"\s(?P<user>[^\s]+)" +
16          r"\s(from)" +
17          r"\s(?P<ip>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})" +
18          r".*$", log_record)
19      r = log_data.groupdict()
20      r['ts'] = dt.strptime(r['ts'], r"%b %d %H:%M:%S")
21      r['ts'] = r['ts'].replace(year=dt.utcnow().year)
22      r['country'] = geolite2.lookup(r['ip']).country
23      return r
24  def exportAuthActivity(path="logs/auth.log"):
25      es = Elasticsearch("http://localhost:9200")
26      log_file = openlogfile(path)
27      for log_record in log_file:
28          try:
29              log_data = parseAuth(log_record)
30              es.index(index="auth", document=log_data)
31          except:
32              pass
33  if __name__ == "__main__":
34      exportAuthActivity("logs/auth.log")
```

如果在 PyCharm 中成功地执行了 import_auth. py 源代码文件，则会将所有 auth. log 日志文件中的记录导入 Elasticsearch。此时，通过 Kibana 中 Discovery 工具的 Create data view 页面能够查看名为 auth 的索引数据，如图 5-29 所示。

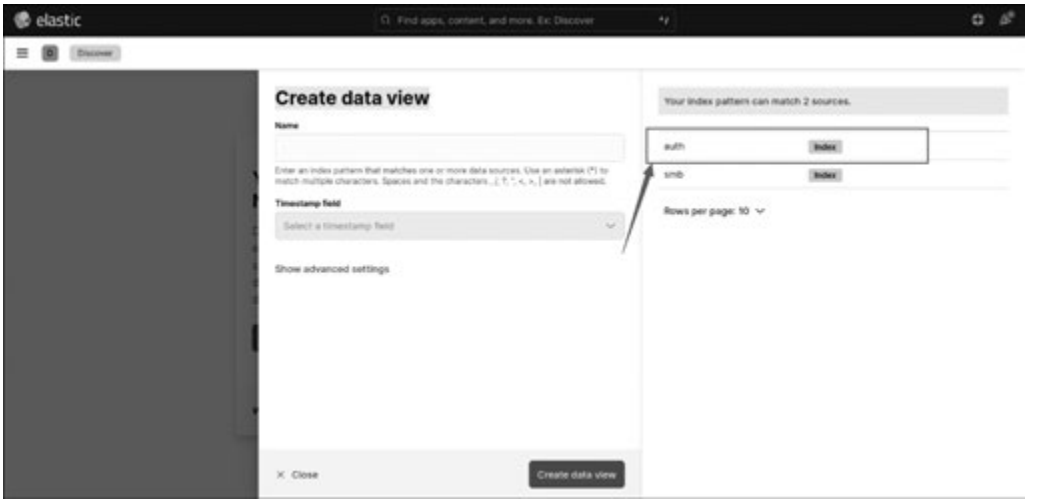

图 5-29　查看索引数据

接下来，通过在 Name 输入框中填写正则表达式来将索引匹配到视图中。同时，用户也能够在 Timestamp field 下拉列表中选择用于标识时间的字段。如果 Name 中的正则表达式成功地匹配到相应索引，则会输出 Your index pattern matches 1 source 提示信息，如图 5-30 所示。

图 5-30　成功匹配 auth 索引

接下来，单击 Create data view 按钮即可完成创建索引任务，并且会自动跳转到 Discover 工具的分析页面。由于 Discover 工具默认显示 15min 内的记录，因此用户需要修改时间范

围来正确显示相应的日志记录。关于修改时间范围的方法，读者可以参考本书 5.2.3 节所介绍的内容来设定合适的范围。如果选择合适的时间范围，则会在 Discover 页面中显示所有的记录信息，如图 5-31 所示。

图 5-31　成功显示所有记录信息

接下来，在 Filter your data using KQL syntax 输入框中填写 action.keyword:"Accepted"作为查询条件便可筛选所有包含成功登录系统的日志记录，如图 5-32 所示。

图 5-32　成功筛选符合条件的日志记录

当然，Kibana 工具同样支持多种其他功能，感兴趣的读者也可以自行查阅资料学习并掌握更多关于 Kibana 和 Elasticsearch 的相关内容。