**湘潭大学**

**软件设计说明书**

**题 目：\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*设计与实现**

|  |  |
| --- | --- |
| **学 院：** | **计算机学院** |
| **专 业：** | **网络空间安全** |
| **组 长：** | **2022123456 张三** |
| **组 员：** |  |
| **组 员：** |  |
| **组 员：** |  |
| **组 员：** |  |
| **指导教师：** |  |

**完成日期： 2025 年 8 月**

目 录

[摘要 I](#_Toc18578)

[Abstract II](#_Toc8504)

[第1章 引言 1](#_Toc9282)

[1.1 设计背景与意义 1](#_Toc14246)

[1.2 国内外研究发展现状 1](#_Toc7589)

[1.3 研究的目标和内容 2](#_Toc27157)

[1.3.1 研究目标 2](#_Toc11433)

[1.3.2 研究内容 2](#_Toc17332)

[1.4 研究思路 2](#_Toc7918)

[1.5 本文的章节安排 3](#_Toc13858)

[1.6 项目分工情况 3](#_Toc6158)

[第2章 相关技术 4](#_Toc21794)

[2.1 开发框架 4](#_Toc10509)

[2.1.1 SpringCloud 4](#_Toc12494)

[2.1.2 Flask 4](#_Toc24626)

[2.2 系统部署 4](#_Toc19069)

[2.2.1 Docker 4](#_Toc31975)

[2.2.2 Nginx 5](#_Toc16823)

[2.2.3 Gunicorn 5](#_Toc3917)

[2.2.4 Gevent 5](#_Toc491)

[第3章 需求分析 6](#_Toc18847)

[3.1 系统概述 6](#_Toc16075)

[3.2 系统功能需求分析 6](#_Toc14757)

[3.2.1 数据库API服务功能需求 6](#_Toc15469)

[3.2.2 数据展示系统功能需求 8](#_Toc4802)

[3.3 系统数据迁移同步 9](#_Toc2476)

[第4章 系统设计 10](#_Toc13662)

[4.1 系统总体设计 10](#_Toc18017)

[4.2 系统详细设计 11](#_Toc18113)

[4.2.1 数据库API服务相关功能设计 11](#_Toc7257)

[4.2.2 数据展示后端相关功能设计 11](#_Toc1172)

[4.3 系统数据库设计 13](#_Toc2379)

[第5章 系统实现 16](#_Toc6565)

[5.1 系统文件结构 16](#_Toc18700)

[5.1.1 数据库API服务文件结构 16](#_Toc9639)

[5.1.2 数据展示后端文件结构 16](#_Toc27348)

[5.2 相关功能模块实现 17](#_Toc30718)

[5.2.1 数据库API服务相关功能实现 17](#_Toc12638)

[5.2.2 数据展示后端相关功能实现 23](#_Toc15830)

[5.3 系统难点及实现 29](#_Toc7443)

[5.3.1 数据库迁移同步 29](#_Toc9756)

[5.3.2 微服务远程调用 30](#_Toc14513)

[第6章 系统功能测试 31](#_Toc21008)

[6.1 系统测试环境 31](#_Toc17420)

[6.2 数据库API服务功能测试 31](#_Toc24450)

[6.3 数据展示后端功能测试 32](#_Toc30159)

[第7章 总结与展望 33](#_Toc518)

[7.1 总结 33](#_Toc5238)

[7.2 展望 33](#_Toc2712)

[参考文献 34](#_Toc26575)

\*\*\*\*\*\*\*\*\*设计与实现

摘要：随着互联网的发展，暗网因为其独特的匿名性已经成为了一些非法信息和违禁物品的交易平台，如毒品交易、枪支交易、信息交易等。暗网上的特点是往来隐蔽、监控困难，正是这些特点使得犯罪在这里滋生，对我国国家安全造成极大的威胁。为了维护国家和社会稳定，亟需一个能够对暗网信息和节点进行监测的系统，帮助相关部门准确地获取网络犯罪线索及电子证据，快速破获网络犯罪。因此，本文针对暗网的特点进行深入研究，设计实现了一个暗网信息监测系统，辅助技术人员对暗网进行监控。本系统采用了Tor代理、爬虫、selenium等技术实现了暗网信息采集、暗网信息监测、暗网信息分析与展示等功能。对于使用用户来说，可以很容易的通过系统获取暗网数据、暗网节点信息，通过系统提供的可视化图表，快速挖掘有效情报。本系统可以有效地对暗网进行监测和采集，为维护国家网络空间安全提供了有力的技术支持。

**关键字：**暗网;数据采集;网络爬虫;信息监测

\*\*\*\*\*\*\*\*\*\*\* Design and Implementation

Abstract: With the development of the Internet, the dark web has become a trading platform for some illegal information and prohibited items due to its unique anonymity, such as drug transactions, gun transactions, information transactions, etc. The dark web is characterized by concealment and difficulty in monitoring. It is these characteristics that make crimes breed here and pose a great threat to my country's national security. In order to maintain national and social stability, there is an urgent need for a system that can monitor dark web information and nodes, help relevant departments to accurately obtain clues and electronic evidence of cyber crimes, and quickly crack down on cyber crimes. Therefore, this paper conducts in-depth research on the characteristics of the dark web, and designs and implements a dark web information monitoring system to assist technical personnel in monitoring the dark web. This system uses Tor proxy, crawler, selenium and other technologies to realize the functions of dark web information collection, dark web information monitoring, dark web information analysis and display, etc. For users, it is easy to obtain dark network data and dark network node information through the system, and quickly mine effective intelligence through the visual charts provided by the system. This system can effectively monitor and collect the dark web, and provides strong technical support for maintaining national cyberspace security.

**Key words:** dark web; data acquisition; web crawler; information monitoring

1. 引言
   1. 设计背景与意义

随着科技的发展，互联网已经融入到现代每个人的生活之中，大多数人接触到的互联网是互联网的第一“层”——表面网络，这是能够被搜索引擎索引到的区域，如各类新闻、购物网站，仅约占整个互联网的4%，剩下的部分全部被深层网络所占据，这是不能够被百度、360搜索等搜索引擎搜索到的部分，比如个人邮件系统、公司内部管理系统都属于深网。而暗网是深网的一个子集，其不仅无法被常见的搜索引擎索引而且无法通过普通的谷歌浏览器或Safari浏览器访问。暗网的运作原则是“洋葱路由”，接入暗网的计算机会通过多个网络之间的跳转，以隐藏网络请求最初来自哪台计算机，从而实现网络的匿名性。这个概念最初是由美国海军提出的，一开始只是作为保护美国在线情报的一种手段，后来在1997年公开了Tor项目并部署到公共领域供任何人使用。

近几年，暗网由于其高度匿名性的特点，被不法分子利用进行违法犯罪活动，暗网中充斥着大量军火、毒品、个人隐私信息等非法交易信息。随着暗网进入门槛降低，以及加密数字货币的普及，国内暗网用户、中文交易网站数量激增，大量毒品、公民个人信息等在暗网交易市场公开交易，例如，2018年8月，当时最大的中文暗网交易市场“暗网中文论坛”中出现一则出售华住酒店数据的帖子，泄漏数据总计近5亿条，涉及1.3亿人，包括姓名、身份证号码、入住时间、离开时间、房间号、消费金额等信息，被标价为8比特币 (当时约合人民币38万元)出售。暗网对我国的影响愈发严重，因此亟需一个可以对暗网数据进行收集、监控的分析系统，为相关部门提供高效情报发现、关联分析和挖掘服务，帮助各执法部门准确地获取网络犯罪线索，快速破获网络犯罪，以降低暗网对国家和社会造成的安全隐患。

* 1. 国内外研究发展现状

针对暗网中非法信息泛滥的问题，国内外的专家学者已经开始着手对暗网信息采集及监控的相关技术展开研究。汤艳君等人在分析暗网匿名通信系统Tor技术原理基础上,设计了一套基于Selenium的暗网爬虫,通过定义的流程自动化地采集暗网网页的数据，为暗网数据采集提供了一定的借鉴意义[1]。于浩佳等人通过Polipo与Tor相结合的方式，优化了爬虫接入Tor网络的速度，显著提高了信息采集效率，监管暗网提出了一种可行的技术手段[2]。国外的Cafarella M J等人开发了一种名为WISE-Integrator的工具,其可以对暗网网页进行自动化文本提取，

并根据内容进行分类[3]。2016年, Eric Nunes等人构建了一个专门用于从以暗网为主的网站手机网络威胁情报的操作系统，同时利用机器学习和自然语言处理技术，对收集到的情报进行分类与分析[4]。上述研究对于本文有着很大的借鉴意义，但随着互联网技术的快速发展，上述研究中所用到的一些技术已经过时或失效，同时也存在着功能缺失的问题，面对暗网中越来越泛滥的违法犯罪行为，对暗网信息监测分析的研究有着十分重要的现实意义。

* 1. 研究的目标和内容
     1. 研究目标

本文的研究目标是提出一种以Tor网络、爬虫、数据可视化为核心的面向暗网的资源采集和数据分析方法，实现对暗网节点进行大规模指纹扫描，辅助技术人员对暗网进行监控，维护国家网络空间安全。

* + 1. 研究内容

针对庞大的暗网空间，提供暗网站点发现功能；对于已发现的站点，扫描该站点的指纹信息并利用爬虫爬取站点信息；将采集到的信息利用可视化工具，通过柱状图、折线图、词云等形式进行展示，为使用者提供高效的情报发现、关联分析和挖掘服务。本文主要针对暗网信息监测分析系统的后端设计与实现，主要研究内容如下：

1. 为暗网信息监测分析系统的数据采集子系统提供信息存取的数据库API服务，对数据进行统一管理
2. 对采集到的数据进行清洗与分析
3. 为前端页面提供数据展示的后端API，前端根据数据构建可视化图标
   1. 研究思路

整个暗网监测分析系统分为三大部分：信息采集子系统、后端数据处理模块、前端展示模块。

信息采集子系统包括暗网站点发现和采集、暗网搜索引擎聚合、爬虫调度框架等功能。该部分开发框架使用基于Python的Flask框架，为了开发的方便数据库采用MongoDB数据库，数据以BSON进行存储，根据域名进行分库来实现不同市场的区分，以接口形式为业务系统提供采集的原始数据。

系统后端数据处理模块需要为爬虫提供数据库存储操作接口，免去爬虫框架频繁连接数据库的操作，同时需要对爬虫采集到的数据进行清洗、处理，以适当的格式返回给前端展示模块。该部分以基于SpringCloud的微服务模式构建。

系统前端展示模块需要将后端返回的JSON格式化数据利用折线图、柱状图等方式进行可视化展示，能够对暗网总体态势和某个暗网节点的信息进行直观展示，方便使用者快速了解实时暗网信息。主要采用的开发框架与工具有：Vue、Ant Design组件库、Echarts数据可视化图表库。

在项目开发完成后通过Docker以容器方式部署至服务器，保证系统的稳定性与可移植性。

* 1. 本文的章节安排

本文介绍了暗网信息监测系统的后端设计与实现，各章节的主要内容如下：

第1章：引言。本章主要介绍了暗网的发展背景和技术原理，阐述了开发暗网信息监测系统的必要性和对于国家安全的意义，收集当前国内外对于暗网的研究现状和研究成果信息，并对本文的研究目标、内容、思路进行叙述。

第2章：相关技术介绍。本章对开发暗网信息监测系统所采用的开发框架与技术进行简要概述。

第3章：需求分析。本章对暗网信息监测系统进行概述并对所需功能进行需求分析。

第4章：系统设计。本章详细介绍了数据采集模块提供服务的数据库API和为前端提供数据的后端数据API的设计思路和过程。

第5章：系统实现。本章对暗网信息监测系统的各部分功能的实现进行详细概述。

第6章：系统功能测试。本章介绍了系统测试环境，并在测试环境下对各目标功能进行测试。

第7章：总结与展望。本章对本文设计和完成的暗网信息监测系统开发过程、研究结果进行总结，并对系统的可改进之处提出展望。

* 1. 项目分工情况

1. 相关技术

本文所设计的暗网信息监测系统前端采用了Vue框架、AntDesign Vue 组件库、Echarts数据可视化图表库，后端采用了SpringCloud框架，NACOS作为微服务的注册中心和配置中心，SpringCloud GateWay作为微服务的网关，基于前后端分离模式开发，开发过程中使用Swagger自动生成接口文档。爬虫数据存储API使用Python Flask框架开发。根据业务的需要，数据库同时使用了MySQL与MongoDB。在部署本暗网信息监测系统的过程中使用到了Nginx、Docker、Gunicorn等技术。本章将介绍SpringCloud、Flask、Docker、Nginx、Gunicorn等技术的相关背景。

* 1. 开发框架
     1. SpringCloud

SpringCloud是分布式微服务架构的一站式解决方案，开发者可以基于SpringCloud提高的一系列框架利用SpringBoot轻松实现微服务系统基础设施的快速构建与部署，SpringCloud提供了服务注册、服务配置、服务网关、负载均衡、分布式消息等一系列分布式系统所需组件，只需要简单的引入与配置即可使用。

* + 1. Flask

Flask是一个轻量级的Python web应用框架，同时也是Python中最受欢迎的web框架之一，其特点是轻巧、简洁、高效、启动速度快、可定制性高，开发者可以通过Flask快速而简单的构建一个Web项目，也能通过引入插件逐渐扩展到复杂的Web项目。Flask没有强制任何依赖或项目结构，开发者可以根据需要自己选择数据库操作工具、数据验证工具或其他组件，给开发者提供了极大的自由度。

* 1. 系统部署
     1. Docker

Docker是一个开源的应用容器引擎，用于快速部署、运行项目。开发者可以构建自己的项目容器并打包成可移植镜像共享给任何人。Docker与传统的虚拟机有着很大区别，Docker中的所有容器以沙箱机制共享宿主机的操作系统，无需为每个容器单独创建操作系统，在共享的同时不同容器间不会相互影响，极大的节省了系统资源、提高运行速度。使用Docker部署项目，可以像运行一个应用程序一样轻松容易。

* + 1. Nginx

Nginx是一款基于C语言开发的高性能、轻量级的Web服务器，因为其稳定性好、系统资源占用少、支持高并发、配置简单等优越特性被广泛运用于各大Web网站项目中。Nginx也支持反向代理、负载均衡、邮件代理服务等功能。

* + 1. Gunicorn

Gunicorn是一个Python的WSGI HTTP服务器，用来解析HTTP请求的网关服务。它可以与web应用（如Django或Flask）配合使用，通常位于反向代理（如Nginx）或负载均衡（如AWS ELB）之后。Gunicorn有多种运行模式，可以根据不同的业务需要选择同步或异步的方式处理请求。Gunicorn可以通过命令行或配置文件来指定各种参数，如绑定的IP和端口，工作进程数，日志文件等。

* + 1. Gevent

Gevent是一个基于Python协程的网络库，支持多种网络协议，比如UDP、TCP、HTTP等，提供了队列、事件、锁等原语。Gevent可以使Python开发者轻松地编写一个高性能网络应用，通过Gevent提供的API可以将应用中的同步的阻塞IO变为非阻塞的异步代码，在极大的提高并发性能的同时还能显著节省内存利用率。

1. 需求分析
   1. 系统概述

随互联网技术的高速发展普及，暗网因为其特有的匿名性和不易追踪性，被很多不法分子利用来作为进行非法交易和违法犯罪活动的场所，对社会稳定和国家安全造成极大的威胁，因此，国家有关部门亟需一种可以方便快捷对暗网进行监控的工具，为业务人员提供高效情报发现、关联分析和挖掘服务，帮助各执法部门准确地掌握网络犯罪线索。本系统从此需求出发，通过爬虫技术对目标站点的信息进行爬取，并使用暗网站点扫描工具对暗网站点进行扫描，实时监控目标站点的状态，帮助相关部门快速掌握线索。对采集到的网站数据进行分析，运用词云、柱状图、饼状图、折线图等形式对数据进行可视化的直观展示，剖析数据的内部联系和潜在价值。

该暗网监测系统后端部分主要需要实现的功能有：提供一个统一的数据库存储接口使得爬虫能够通过调用接口将爬取到的数据存储到MongoDB数据库中，避免在爬虫程序中加入操作数据库的代码，达到不同网站爬虫数据存储格式的统一且简化爬虫程序的开发的效果；其次，需要将爬虫采集到的数据进行清洗后处理成前端需要的格式，便于前端调用数据进行可视化展示；最后，需要对暗网搜索引擎做一个聚合，使得使用者无需配置Tor代理也可进行暗网站点的搜索。

* 1. 系统功能需求分析
     1. 数据库API服务功能需求

该模块主要是为爬虫提供一个统一的数据库存取接口，考虑到数据需要定期与关系型数据库MySQL同步，所以需要设计一个统一的存储数据格式。暗网信息监测系统采集的数据主要来自于中文暗网交易市场，通过对交易市场页面结构分析，提取出它们共有的部分，如：市场名称、商品描述、商品价格、交易评价等，设计出针对MongoDB数据库的统一JSON格式的商品数据结构，如下所示：

{

  "\_id": "42e4cadb-843e-4955-ada0-f577c1e43361",    //自动生成的UUID

  "link": "http://xxxxx.onion/?product=xxxx", //商品网址

  "get\_time": 154265478951, //商品入库时间

  "product\_name" : "xxx", //商品名称

  "product\_category": "xxx", //商品分类

  "product\_description": "xxx", //商品描述

  "publisher": { //商品发布者信息

    "nick\_name": "发布者昵称", //商品发布者昵称

    "total\_revenue": 2.089, //商品发布者总收入

    "last\_login\_time": 1524659847854, //商品发布者最后登录

    "registration\_time": 15426587491244 //商品发布者注册时间

  },

  "pub\_time": 1354265987412, //商品发布时间

  "product\_price": "0.00114比特币", //商品价格

  "status": "在售", //商品状态

  "sold": 99, //商品已售数

  "inventory": 200, //商品库存

  "market": "茶马古道", //商品所属交易市场

  "snapshot": "http://xxxx:8000/onion/5e86a0298.jpg", //商品快照地址

  "comments": [{//评论信息

    "comment\_user": "yangcong",//评论者ID

    "comment\_date": "158678935475",//评论时间

    "comment\_text": "你要做什么产品？"//评论内容

  }],

  "transactions": [{//交易信息

    "Buyer": "买家ID",

    "purchase\_date": "1354265987356" //交易时间

  }]

}

还需要对暗网站点进行指纹扫描，建立站点指纹数据库，指纹信息包括站点开启的端口、支持的服务等，如是否开启ftp服务、smtp服务、tls加密等信息，同时还有站点使用的加密货币类型，基于以上需求，设计了站点指纹JSON数据格式，如下所示：

{

  "hiddenService": "alibaba2kw6qoh6o.onion",

  "dateScanned": "2023-04-07T14:51:05.612114913Z",

  "online": false,

  "webDetected": true,

  "tlsDetected": true,

  "sshDetected": true,

  "ftpDetected": true,

  "smtpDetected": true,

  "bitcoinDetected": false,

  "mongodbDetected": true,

  "vncDetected": true,

  "xmppDetected": false,

  "skynetDetected": false,

  "bitcoinServices": {

    "bitcoin": {

      "detected": true,

      "userAgent": "",

      "prototocolVersion": 0,

      "onionPeers": ""

},

"dogecoin": {

      "detected": false,

      "userAgent": "",

      "prototocolVersion": 0,

      "onionPeers": ""

    },

    "litecoin": {

      "detected": false,

      "userAgent": "",

      "prototocolVersion": 0,

      "onionPeers": ""

    }

  }

}

* + 1. 数据展示系统功能需求

经过分析，对于暗网信息监控的前端页面主要有：态势总览、暗网交易、暗网节点、暗网溯源、暗网站点分类、暗网搜索引擎等，后端需要为前端页面中的功能提供数据接口，前端只需要调用接口获取数据并将数据展示在网页上，而无需关心数据的来源和格式，将后端需要实现的功能进行分类汇总，如图3-1所示。



图 3‑1 各模块功能图

* 1. 系统数据迁移同步

根据系统功能特点，信息采集模块使用Python语言进行开发，为了方便数据库连接及存储使用了MongoDB数据库，数据展示模块使用JAVA语言开发并采用MySQL数据库，因此需要定期将信息采集模块新采集到的数据从MongoDB数据库中迁移至MySQL中以完成数据的同步，数据迁移同步通过脚本文件执行，数据迁移同步的频率可以为一天一次，也可在爬取完一个新站点的数据后立即进行同步。

MongoDB自带的工具可以将数据导出为JSON格式，再通过编写脚本读取JSON数据，构造动态SQL语句并提交MySQL中执行即可将数据插入至MySQL数据库的指定数据表中，所以在MySQL中建立商品表时需要保持表结构与MongoDB中数据格式的一致以方便数据迁移同步。

1. 系统设计
   1. 系统总体设计

该暗网信息监测系统的总体设计架构如图4-1所示。

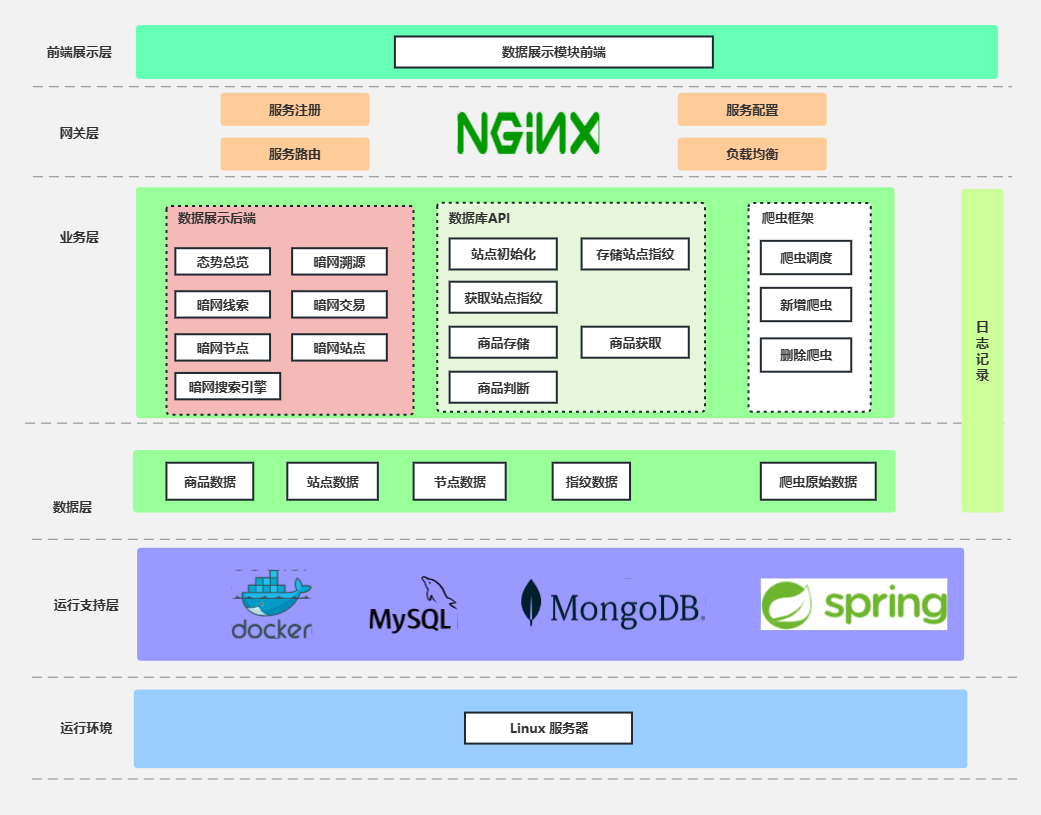


图 4‑1 系统总体设计图

数据库API服务是为了简化数据采集子模块的开发而设计的，在数据采集子模块采集完数据后无需关心如何连接数据库、怎样存储数据、以什么样的格式存储数据，只需要参照数据库API接口文档，调用相应的接口，即可完成数据的存取。数据库API服务的业务流程如图4-2所示。

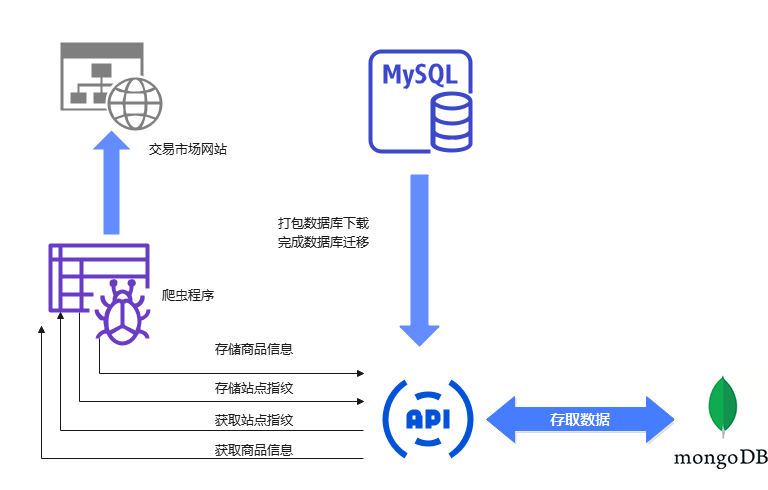


图 4‑2 数据库API服务业务流程设计图

数据展示系统后端是为前端信息展示、渲染图表、分类查询等功能提供数据支撑，前端使用POST或GET方式向后端相应接口发送请求并携带参数，后端接收到参数后会根据条件构造查询语句在数据库中进行查询，并根据前端需要的格式对数据进行去重、合并、格式化等操作，最终以JSON格式返回给前端。数据展示后端的业务流程如图4-3所示。

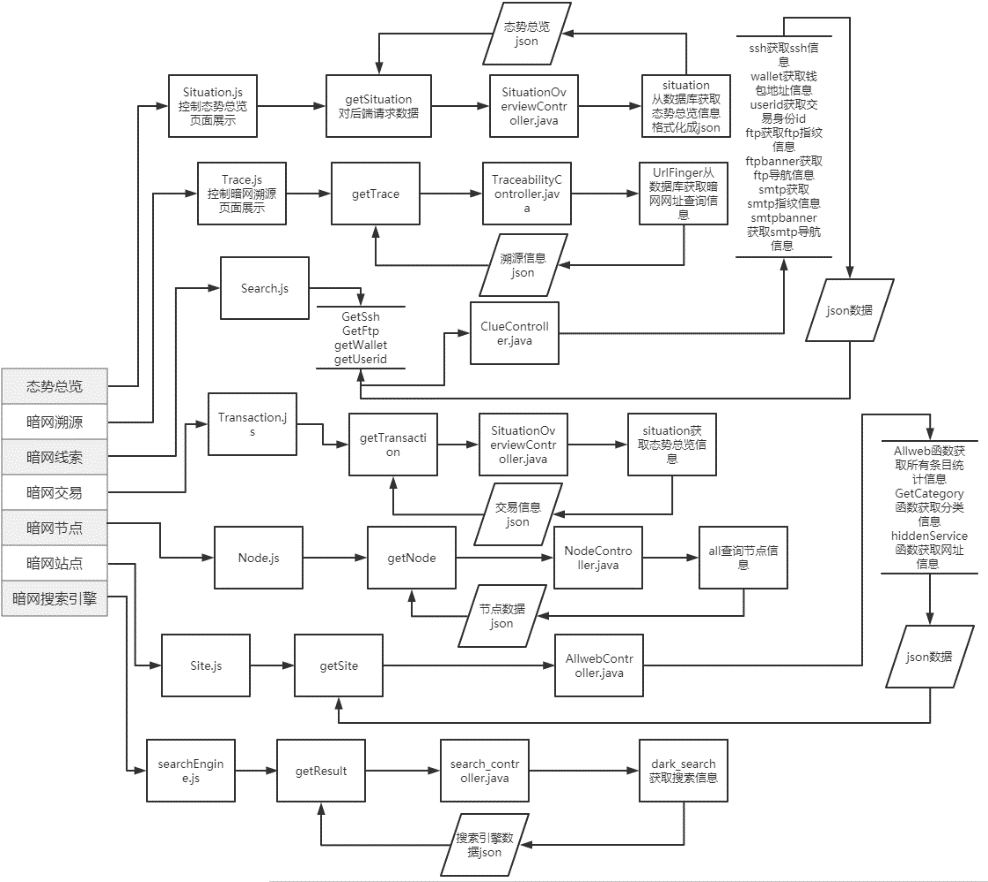


图 4‑3 数据展示系统前后端业务流程设计

* 1. 系统详细设计
     1. 数据库API服务相关功能设计

如果要将商品或站点数据存入数据库，只需要以POST方式请求数据库API的存储接口，并按照接口文档规定的JSON格式携带数据，数据库API在接收到传递来的参数后会负责数据的校验、存储，并将存储结果以布尔值形式返回，爬虫框架只需要根据返回值即可判断本次存储操作是否成功。

如果要获取数据库中的数据，只需要以GET方式发送请求，并携带查询条件，如关键词、入库时间等，数据库API会根据条件查询数据库并返回结果。

数据库API还提供能够将数据从MongoDB数据库迁移至MySQL数据库的接口，可将MongoDB中的数据导出为JSON格式或SQL格式，MySQL可以直接读取该类型文件并导入数据。

* + 1. 数据展示后端相关功能设计

数据展示后端基于微服务开发，根据业务逻辑拆分为多个独立的模块，采用Spring Cloud Alibaba生态：Nacos做为服务注册中心、配置管理中心，提供服务发现、动态配置管理的功能；SpringCloud-Gateway做为API网关，所有的请求都通过统一的网关与微服务进行交互，由API网关进行请求分发，减少了客户端与各个微服务之间的交互次数同时也可实现负载均衡；SpringCloud-Feign做为远程调用工具，实现不同微服务间的相互调用。

主要分为darkweb-allweb、darkweb-node、darkweb-search、darkweb-transacation、darkweb-gateway等服务。

darkweb-allweb提供网站指纹相关接口，如根据网址从数据库查询对应站点的指纹数据返回给前端、判断对应站点是否开启某些端口等。

darkweb-node提供暗网节点相关接口，如根据提供的节点类型，查询该类型的全部节点数据。

darkweb-search是暗网搜索引擎相关接口，根据关键词进行搜索，将结果返回给前端。

darkweb-transacation提供商品交易相关接口，从数据库获取暗网交易页面所需数据返回前端。

darkweb-gateway为所有微服务提供一个统一的入口，使用者无需关心具体服务的地址，只需将请求发给网关，由网关负责根据请求地址将请求转发到相应的微服务，同时还提供权限校验、负载均衡等功能。

数据展示后端系统设计如图4-4所示。

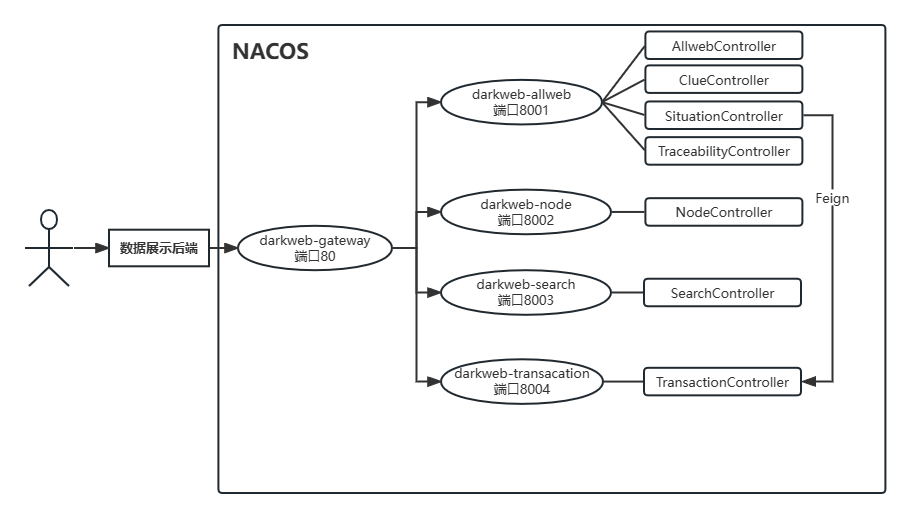


图 4‑4 数据展示后端系统设计图

* 1. 系统数据库设计

根据项目功能以及数据库设计规范，对相关数据表进行设计：表4-1为Site表，用于存储数据采集子系统已爬取或待爬取的网站列表，包括网站的唯一ID、网站的地址字段； 表4-2为node表，用于存储已经发现的暗网节点详细信息，包括节点网络类型、节点公钥、节点分类、发现时间等字段；表4-3为fingerprint，用于存储由数据采集子模块通过OnionScan工具扫描到的网站指纹信息，包括网站是否在线、支持的加密数字货币类型、端口开放信息等字段；表4-4为product表，用于存储交易市场中的所有商品信息，包括商品名称、商品描述、交易评价等字段。

表 4-1 Site表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 字段类型 | 长度 | 是否为空 |
| Id | varchar(200) | varchar | 200 | NO |
| site\_name | varchar(500) | varchar | 500 | NO |

表 4-2 node表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 字段类型 | 长度 | 是否为空 |
| Id | int(255) | int | 255 | NO |
| node\_type | varchar(500) | varchar | 255 | NO |
| pubkey | varchar(255) | varchar | 255 | NO |
| signkey | varchar(255) | varchar | 255 | NO |
| NetId | varchar(255) | varchar | 255 | NO |
| Caps | varchar(255) | varchar | 255 | NO |
| Version | varchar(255) | varchar | 255 | NO |
| Signature\_Type | varchar(255) | varchar | 255 | NO |
| peerSize | varchar(255) | varchar | 255 | NO |
| crypto\_Type | varchar(255) | varchar | 255 | NO |
| get\_time | date | date |  | NO |

表 4-3 fingerprint表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 字段类型 | 长度 | 是否为空 |
| hiddenService | varchar(200) | varchar | 200 | NO |
| netType | varchar(10) | varchar | 10 | NO |
| dateScanned | bigint(20) | bigint |  | YES |
| first\_category | varchar(20) | varchar | 20 | NO |
| second\_category | varchar(20) | varchar | 20 | YES |

续表 4-3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| online | tinyint(1) | tinyint |  | YES |
| performedScans | varchar(1000) | varchar | 1000 | YES |
| webDetected | tinyint(1) | tinyint |  | YES |
| tlsDetected | tinyint(1) | tinyint |  | YES |
| sshDetected | tinyint(1) | tinyint |  | YES |
| ricochetDetected | tinyint(1) | tinyint |  | YES |
| ircDetected | tinyint(1) | tinyint |  | YES |
| ftpDetected | tinyint(1) | tinyint |  | YES |
| smtpDetected | tinyint(1) | tinyint |  | YES |
| bitcoinDetected | tinyint(1) | tinyint |  | YES |
| mongodbDetected | tinyint(1) | tinyint |  | YES |
| vncDetected | tinyint(1) | tinyint |  | YES |
| xmppDetected | tinyint(1) | tinyint |  | YES |
| skynetDetected | tinyint(1) | tinyint |  | YES |
| crawls | varchar(1000) | varchar | 1000 | YES |
| pgpKeys | varchar(1000) | varchar | 1000 | YES |
| certificates | varchar(1000) | varchar | 1000 | YES |
| bitcoinServices | varchar(2000) | varchar | 2000 | YES |
| sshKey | varchar(1000) | varchar | 1000 | YES |
| sshBanner | varchar(1000) | varchar | 1000 | YES |
| ftpFingerprint | varchar(1000) | varchar | 1000 | YES |
| ftpBanner | varchar(1000) | varchar | 1000 | YES |
| smtpFingerprint | varchar(1000) | varchar | 1000 | YES |
| smtpBanner | varchar(1000) | varchar | 1000 | YES |
| lastAction | varchar(1000) | varchar | 1000 | YES |
| timedOut | tinyint(1) | tinyint |  | YES |
| error | varchar(1000) | varchar | 1000 | YES |
| identifierReport | varchar(1000) | varchar | 1000 | YES |
| simpleReport | varchar(1000) | varchar | 1000 | YES |
| hidden\_service | varchar(255) | varchar | 255 | NO |
| bitcoin\_detected | bit(1) | bit |  | YES |
| bitcoin\_services | varchar(255) | varchar | 255 | YES |
| date\_scanned | bigint(20) | bigint |  | YES |
| ftp\_banner | varchar(255) | varchar | 255 | YES |
| ftp\_detected | bit(1) | bit |  | YES |
| ftp\_fingerprint | varchar(255) | varchar | 255 | YES |
| identifier\_report | varchar(255) | varchar | 255 | YES |
| irc\_detected | bit(1) | bit |  | YES |
| last\_action | varchar(255) | varchar | 255 | YES |
| mongodb\_detected | bit(1) | bit |  | YES |
| performed\_scans | bit(1) | bit |  | YES |

续表 4-3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| pgp\_keys | varchar(255) | varchar | 255 | YES |
| ricochet\_detected | bit(1) | bit |  | YES |
| simple\_report | varchar(255) | varchar | 255 | YES |
| smtp\_detected | bit(1) | bit |  | YES |
| skynet\_detected | bit(1) | bit |  | YES |
| smtp\_banner | varchar(255) | varchar | 255 | YES |
| smtp\_fingerprint | varchar(255) | varchar | 255 | YES |
| ssh\_banner | varchar(255) | varchar | 255 | YES |
| ssh\_detected | bit(1) | bit |  | YES |
| ssh\_key | varchar(255) | varchar | 255 | YES |
| timed\_out | bit(1) | bit |  | YES |
| tls\_detected | bit(1) | bit |  | YES |
| vnc\_detected | bit(1) | bit |  | YES |
| web\_detected | bit(1) | bit |  | YES |
| xmpp\_detected | bit(1) | bit |  | YES |
| port | int(10) | int |  | NO |
| language | varchar(10) | varchar | 10 | NO |
| container | varchar(10) | varchar | 10 | NO |

表 4-4 product表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 字段类型 | 长度 | 是否为空 |
| id | varchar(200) | varchar | 200 | NO |
| link | varchar(1000) | varchar | 1000 | NO |
| get\_time | bigint(13) | bigint |  | NO |
| product\_name | varchar(500) | varchar | 500 | NO |
| product\_category | varchar(100) | varchar | 100 | NO |
| product\_description | varchar(1000) | varchar | 1000 | NO |
| publisher | varchar(1000) | varchar | 1000 | NO |
| pub\_time | bigint(13) | bigint |  | NO |
| product\_price | varchar(30) | varchar | 30 | NO |
| status | varchar(10) | varchar | 10 | NO |
| sold | int(10) | int |  | NO |
| inventory | int(10) | int |  | NO |
| market | varchar(100) | varchar | 100 | NO |
| Snapshot | varchar(500) | varchar | 500 | NO |
| comments | varchar(1000) | varchar | 1000 | NO |
| transactions | LONGTEXT | LONGTEXT |  | NO |
| num | bigint(20) | bigint |  | YES |

1. 系统实现
   1. 系统文件结构
      1. 数据库API服务文件结构

数据库API服务由于功能较为简单所以选择使用Python语言基于Flask轻量级Web框架进行开发，能够显著提升开发效率，降低代码编写复杂度。其项目结构中主要包含项目入口文件、注册蓝图、数据库配置文件、接口文档、工具类等文件，如图5-1所示。

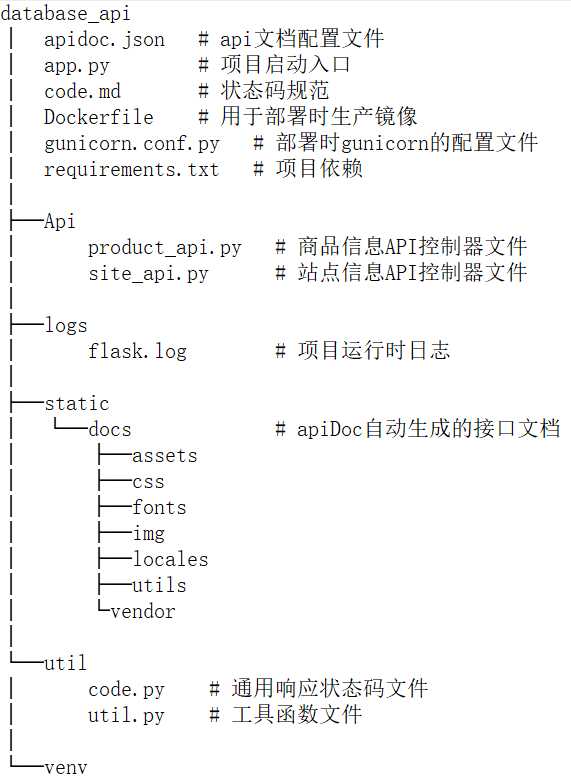


图 5‑1 数据库API服务文件结构树状图

* + 1. 数据展示后端文件结构

数据展示后端基于Springcloud Alibaba，采用微服务模式设计，将整个系统拆分为darkweb-allweb、darkweb-node、darkweb-search、darkweb-transacation、darkweb-gateway等服务，每个服务都基于Springboot进行编写，具体结构如图5-2所示。其中，Dao层负责数据的连接及操作；Service层负责业务功能的具体实现；Controller层负责与前端进行交互，具体的业务逻辑交由Service层进行处理；Entity中每一个类的属性都与数据表的字段一一对应，用于接收Dao层的查询结果；Vo中的类根据前端的需要将Entity中的数据转换为前端便于使用的数据结构；Config中的类为配置类，起到配置的作用。

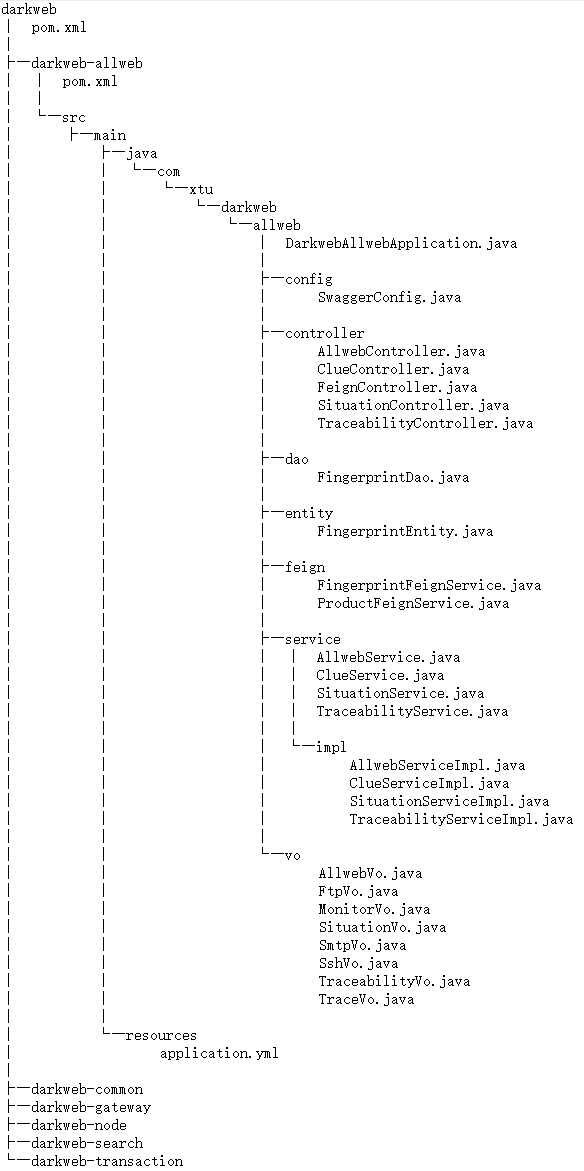


图 5‑2 数据展示后端文件结构树状图

* 1. 相关功能模块实现
     1. 数据库API服务相关功能实现

存储商品信息功能实现。接口的请求方式为POST，对接收到的数据进行校验，首先与系统规定的商品格式比对进行格式校验，若通过格式校验再校验商品对应的站点是否在数据库中进行过初始化，通过校验的参数存入数据库，若未通过校验则返回相应的错误码和提示信息。商品数据格式校验函数与商品存储接口的代码如下所示，API接口文档如图5-3所示。

def cp(product, site): # 检查商品数据格式是否合法  
 # 打开json文件  
 target = os.path.join(app.static\_folder, 'json/product.json')  
 with open(target, "r") as f:  
 # 读取文件内容  
 data = f.read()  
 # 将json字符串转为字典  
 data\_dict = json.loads(data)  
 if cs(site) is None:  
 return None, cs(site)  
 if len(list(mongo.cx["site"]['site\_name'].find({"site\_name": site}))) == 0:  
 return None, False  
 return cj(data\_dict, product), site

@product\_api.route('/save-product', methods=['Post'])  
def save\_product():  
 args = request.args # url链接中的参数  
 form = request.form # 请求体中的参数  
 if "site" not in args or "product" not in form:   
 return msg(103) # 若所需的参数不完整，返回错误信息  
 p\_res, s\_res = cp(form["product"], args["site"]) # 校验参数  
 if not s\_res:  
 return msg(107) # 返回站点未初始化错误  
 if p\_res is None or s\_res is None:  
 return msg(104) # 返回参数格式错误  
 conn = mongo.cx # MongoDB连接实例对象  
 if len(list(conn["product"][s\_res].find({"product\_name": p\_res['product\_name']})))!=0: # 根据商品名称判断商品是否存在  
 return msg(108) # 返回商品已经存在错误  
 conn["product"][s\_res].insert\_one(p\_res) # 在数据库中插入一条记录  
 return msg(100, s\_res)

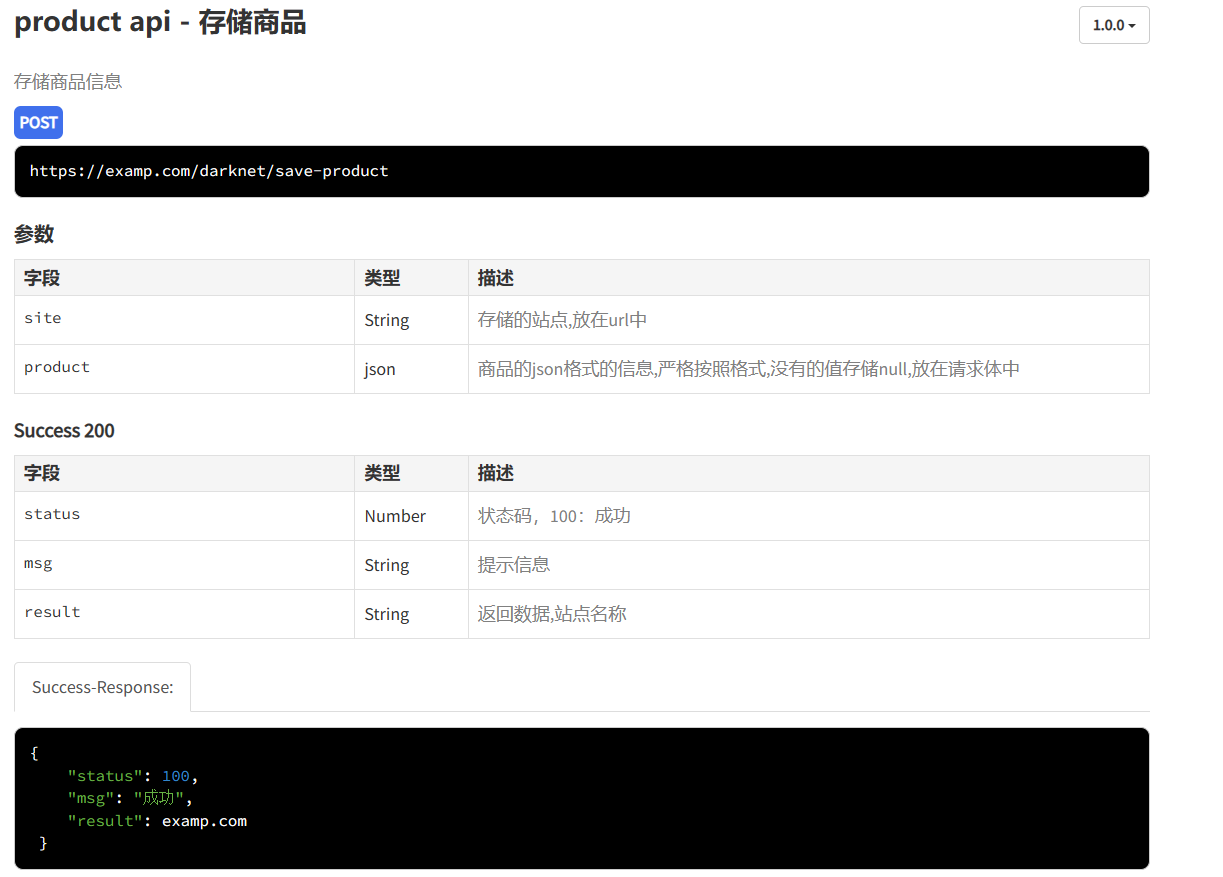


图 5‑3 存储商品信息API文档图

获取商品信息功能实现。接口的请求方式为GET，获取前端请求中的时间参数，通过正则表达式校验该时间参数是否为合法的UNIX时间戳格式，若校验通过将该时间参数转换为字符串格式，并使用该参数构造查询条件在数据库中进行查询，将查询到的所有商品数据返回给前端，时间戳校验函数与获取商品信息接口代码如下所示，API接口文档如图5-4所示。

def ct(str): # 校验时间戳参数  
 rule = "^[0-9]+$"  
 time = re.compile(rule).match(str)  
 res = time.string if time is not None else None  
 return res

@product\_api.route('/getgoods', methods=['Get'])  
def get\_goods():  
 args = request.args  
 if "time" not in args: # 判断参数是否携带  
 return msg(103) # 若所需的参数不完整，返回错误信息  
 time = ct(args["time"]) # 校验参数  
 if time is None:  
 return msg(104) # 返回参数格式错误  
 conn = mongo.cx # 连接实例  
 res = []  
 for item in conn['site']['site\_name'].find():  
 res += list(conn["product"][item["site\_name"]].find({"get\_time": {"$gte": time}})) # 查询到的结果追加到res数组中  
 if not res:  
 return msg(101) # 若res为空，返回无商品数据错误  
 return msg(100, res)



图 5‑4 获取商品信息API文档图

存储网站指纹功能实现。请求方式为POST，接收前端发来的站点参数和网站指纹参数，首先会查询站点是否已经在数据库中初始化，若未初始化会返回相应的错误码和提示信息，接着对网站指纹格式进行校验，若格式不正确或内容不完整返回相应的错误码和提示信息，对于合法请求参数，会在对应站点的数据库中插入该网站指纹记录，网站指纹存储接口代码如下所示，API接口文档如图5-5所示。

@site\_api.route('/save-site-fingerprint', methods=['Post'])  
def save\_site\_fingerprint():  
 args = request.args  
 form = request.form  
 if "site" not in args or "fingerprint" not in form:  
 return msg(103) # 参数不完整  
 fp\_res, s\_res = csf(form["fingerprint"], args["site"]) # 对指纹和站点进行校验  
 if s\_res is None or fp\_res is None:  
 return msg(104) # 参数格式错误  
 if not s\_res: # 站点未初始化  
 return msg(105) # 站点需要经过初始化才能创建指纹  
 if not fp\_res: # 提供的站点与指纹中的站点不一致  
 return msg(109)  
 # upsert=True表示若记录不存在会插入一条记录  
 mongo.cx["site"]["site\_fingerprint"].update\_one({"hiddenService": fp\_res['hiddenService']}, {"$set": fp\_res}, upsert=True)  
 return msg(100, s\_res)



图 5‑5 存储网站指纹API文档图

获取网站指纹API功能实现。请求方法为GET，查询数据库中所有站点的所有网站指纹信息，将查询结果返回给前端，代码如下所示，API接口文档如图5-6所示。

@site\_api.route('/getfinger', methods=['Get'])  
def getfinger():  
 # 使用字典推导式对数组中每个字典中的\_id字段进行str()处理  
 res = [{k: str(v) if k == '\_id' else v for k, v in d.items()} for d in list(mongo.cx["site"]["site\_fingerprint"].find())]  
 return msg(100, res)



图 5‑6 获取网站指纹API文档图

API接口代码编写完成后，还需要部署在Linux服务器上，为了系统的稳定型与可移植性，采用Docker容器方式部署。由于Flask应用只是一个符合WSGI规范的Python应用，不能独立运行，需要依赖其他的组件提供服务器功能，所以选择了Gunicorn作为HTTP 服务器，但Gunicorn 默认使用同步阻塞的网络模型(-k sync)，对于大并发的访问表现不够好，我们选择套一个gevent增加并发量，整体结构如图5-7所示。

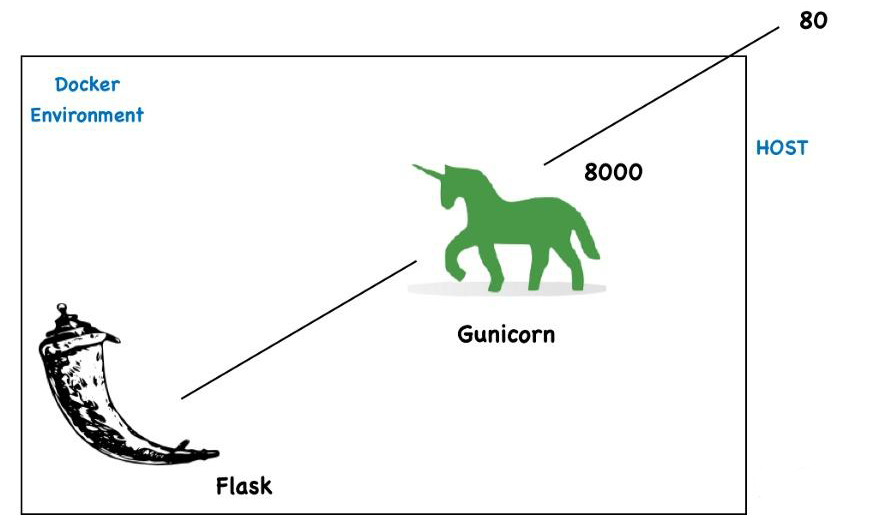


图 5‑7 数据库API服务架构图

创建一个 Dockerfile 文件，以便 Docker 镜像的构建，Dockerfile文件代码如图5-8所示。首先引入Python3.10的环境，第二行 WORKDIR 后面写的是要部署到服务器上的路径，最后一行里面的 start 是我们上面写的 python 启动文件名，app 是启动文件里面要启动的应用名。



图 5‑8 数据库API服务Dockerfile图

在Dockerfile的同级目录下执行如下命令，便可以开始构建Docker镜像：

sudo docker build -t 'dateabase\_api'

镜像生成完成后，以后台方式运行该镜像，将容器的80端口映射到宿主机的5000端口，即可启动数据库API服务，命令如下所示：

sudo docker run -d -p 5000:80 --name database\_api database\_api

* + 1. 数据展示后端相关功能实现

数据展示后端采用Java语言开发，基于SpringCloud微服务，Maven作为项目管理工具，数据库选择轻量级的MySQL数据库。

由于基于微服务模式开发，为了方便管理需要引入服务注册中心与配置中心，以实现服务的发现与管理、配置动态修改等需求，我们选择了阿里巴巴的开源项目Nacos。要将微服务注册到Nacos中，首先修改pom.xml文件，引入Nacos Discovery Starter。

<dependency>  
 <groupId>com.alibaba.cloud</groupId>  
<artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>  
</dependency>

在每个微服务的 /src/main/resources/application.yml 配置文件中配置 Nacos Server 地址。

spring:  
 cloud:  
 nacos:  
 serverAddr: 127.0.0.1:8848  
 discovery:  
 server-addr: 127.0.0.1:8848

在启动类上添加注解，开启服务发现功能。

@EnableDiscoveryClient  
@SpringBootApplication()  
public class DarkwebGatewayApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(DarkwebGatewayApplication.class, args);  
 }  
}

配置完成后，在Nacos后台管理界面可以看到服务列表，有服务的实例数、健康状态等信息，同时可以对服务的上下线进行管理，如图5-9所示。



图 5‑9 Nacos后台管理界面

不同的微服务具有不同的地址与端口，而一个业务可能需要同时调用多个微服务才能完成，若让前端一次调用不同的微服务来完成该业务显然不现实，会存在多次建立连接请求而消耗大量系统资源、微服务重新划分前端需要调整大量代码等问题，所以需要引入API网关，前端使用统一的地址前缀发送请求，所有的请求都会经过API网关，由API网关根据请求地址后缀将请求转发给对应的微服务，前端无需关心业务是如何实现的，所有的业务逻辑都由后端组织微服务相互调用来完成。API网关的具体工作原理如图5-10所示。

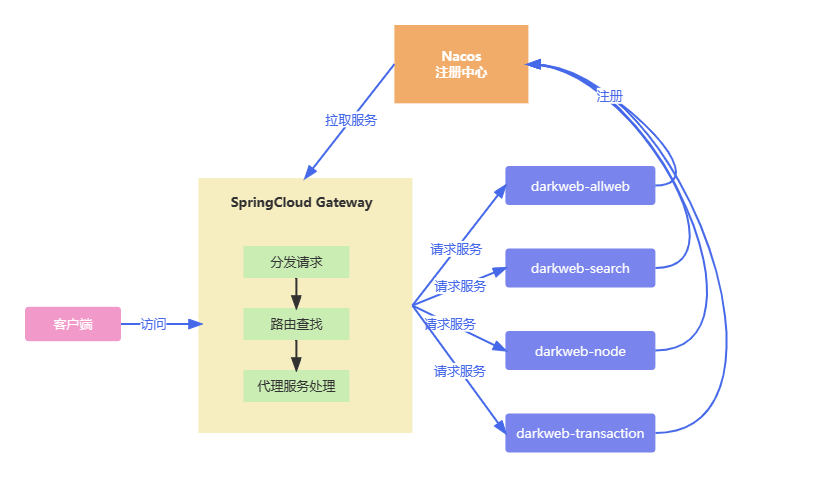


图 5‑10 API网关工作原理图

其中分发请求及路由查找依赖的是HandlerMapping，其根据断言（Predicates）规则来执行相应的路径重写操作，本文的部分路由配置如下所示：

spring:

cloud:

gateway:

routes:

- filters:

- RewritePath=/darkweb/api/?(?<segment>.\*),/${segment}

id: transaction\_route

predicates:

- Path=/darkweb/api/transaction/\*\*

uri: lb://darkweb-transaction

- filters:

- RewritePath=/darkweb/api/?(?<segment>.\*),/${segment}

id: situation\_route

predicates:

- Path=/darkweb/api/situation/\*\*

uri: lb://darkweb-allweb

- filters:

- RewritePath=/darkweb/api/?(?<segment>.\*),/${segment}

id: situation\_route

predicates:

- Path=/darkweb/api/situation/\*\*

uri: lb://darkweb-allweb

- filters:

- RewritePath=/darkweb/api/?(?<segment>.\*),/${segment}

id: traceability\_route

predicates:

- Path=/darkweb/api/Traceability/\*\*

uri: lb://darkweb-allweb

每个微服务使用SpringBoot框架基于MVC架构编写，分为controller层、service层、dao层三层，遵循高内聚低耦合原则，使代码具有更高的可扩展性、可复用性、可维护性和灵活性。dao层是数据持久化层，负责操作数据库，实现持久化层的框架有很多，常用的有JdbcTemplate、JPA、Mybatis等。本文选择基于Mybatis的增强工具Mybatis-Plus，其提供了许多便利的功能，如通用CRUD操作、代码生成器、分页插件等，可以简化开发、提高效率。

以暗网溯源模块的获取溯源信息接口为例，演示系统层级间的相互调用关系。

1. controller层:

该层主要是定义接口地址、请求方式，接收请求的携带的数据，将数据交给service层做进一步的处理。controller层无需关心业务具体的逻辑如何实现，只需要关注数据接收和数据返回。

@RequestMapping("/Traceability")  
@RestController  
public class TraceabilityController {  
 @Autowired  
 TraceabilityService traceabilityService;  
 @Operation(summary = "搜索网址查询",description = "例如?url=shoptwgap2x3xbwy.onion")  
 @GetMapping("/UrlFinger")  
 public R urlFinger(@RequestParam(name = "Url", required = false,defaultValue = "") String url){  
 List<TraceabilityVo> traceabilityVos = traceabilityService.queryUrlFinger(url);  
 return R.ok().put("data", traceabilityVos);  
 }  
}

1. service层：

service层会对controller层传递来的数据进行校验及处理，若数据合法且正确会对数据进行提取，并构造查询条件，调用dao层进行数据库查询，将查询到的结果返回给controller层。

@Service  
public class TraceabilityServiceImpl implements TraceabilityService {  
 @Autowired  
 FingerprintDao fingerprintDao;  
 @Override  
 public List<TraceabilityVo> queryUrlFinger(String url) {  
 QueryWrapper<FingerprintEntity> queryWrapper = new QueryWrapper<>();  
 queryWrapper.eq("hiddenService", url);  
 List<FingerprintEntity> fingerprintEntities = fingerprintDao.selectList(queryWrapper);  
 List<TraceabilityVo> traceabilityVos = fingerprintEntities.stream().map(fingerprintEntity -> {  
 TraceabilityVo traceabilityVo = new TraceabilityVo();  
 BeanUtils.copyProperties(fingerprintEntity, traceabilityVo);  
 return traceabilityVo;  
 }).toList();  
 return traceabilityVos;  
 }  
}

1. dao层

因为我们使用了Mybatis-Plus，Mybatis-Plus封装了通用的CRUD操作，所以对于一些简单的增删改查操作，我们无需编写代码，只需要继承Mybatis-Plus提供的BaseMapper接口，并传入与数据表字段对应的实体类，即可调用dao层的CRUD方法。

@Mapper  
public interface FingerprintDao extends BaseMapper<FingerprintEntity> {}

数据展示后端相关功能全部编写完成后，还需要生成在线API接口文档方便前端查看和调试相关接口。本文通过引入基于Swagger和OpenAPI3的Knife4j作为API接口实时生成工具，具体实现如下：

1. 通过Maven引入依赖，在项目的pom.xml文件的依赖项处添加以下代码引入knife4j

<dependency>

<groupId>com.github.xiaoymin</groupId>

<artifactId>knife4j-openapi3-jakarta-spring-boot-starter</artifactId>

<version>4.1.0</version>

</dependency>

1. 在application.yml中进行相关配置

springdoc:  
 swagger-ui:  
 path: /swagger-ui.html  
 api-docs:  
 path: /v3/api-docs

1. 浏览器访问http://ip:port/doc.html即可查看在线API接口文档，如图5.11所示。



图 5‑11 在线API接口文档

* 1. 系统难点及实现
     1. 数据库迁移同步

根据业务特点，数据采集子系统使用的MongoDB数据库，信息分析展示后端模块采用MySQL数据库，需要定期将采集到的新数据从MongoDB打包迁移至MySQL中以完成数据的同步。

本文通过编写Python脚本连接MongoDB数据库并将MongoDB数据库的指定集合中的所有文档数据导出，通过遍历的方式动态生成一个SQL文件，通过pymysql连接到MySQL数据库，将该文件提交至MySQL执行实现数据的迁移同步，具体代码如下所示：

mongo\_client = 'mongodb://%s:%s@127.0.0.1:27017' % (mongo\_username, mongo\_password)

# 连接MongoDB数据库

client = pymongo.MongoClient(mongo\_client)

db = client["product"]

collection = db["test.onion"]

# 连接MySQL数据库

mysql\_conn = pymysql.connect(host="127.0.0.1", port=3306, user="root", password="xtu@darkweb", database="darkweb")

mysql\_cursor = mysql\_conn.cursor()

# 遍历集合中的每个文档

for doc in collection.find():

# 获取文档中的所有字段和值

fields = list(doc.keys())

values = list(doc.values())

# 生成一条插入语句

sql = "INSERT INTO product (" + ", ".join(fields) + ") VALUES (" + ", ".join(

["'" + str(v) + "'" for v in values]) + ");\n"

# 执行MySQL插入语句

mysql\_cursor.execute(sql)

mysql\_conn.commit()

# 关闭mysql连接

mysql\_conn.close()

mysql\_cursor.close()

* + 1. 微服务远程调用

不同的微服务之间需要相互调用，例如darkweb-allweb服务中的态势总览中一项功能与darkweb-transition服务中的某个接口功能相同，我们可以通过远程调用darkweb-transition服务中相应接口来完成相关业务，避免重复编写代码。

我们使用声明式的 HTTP 客户端OpenFeign来实现微服务间的远程调用。OpenFeign扩展了对 SpringMVC注解的支持，同时整合了负载均衡和服务熔断功能。我们只需要编写一个使用注解表明的接口并在注解中指定需要调用的微服务名称，该接口即可与提供远程调用功能的微服务完成绑定，代码如下所示：

1. 引入依赖

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-openfeign</artifactId>  
</dependency>

1. 在启动类上添加注解，开启feign功能

@EnableFeignClients(basePackages = "com.xtu.darkweb.allweb.feign")

1. 创建远程接口，并添加注解

@FeignClient(name = "darkweb-transaction")  
public interface ProductFeignService {  
 @GetMapping("/transaction/feign/product-count")  
 public Long getProductCount();  
}

1. 系统功能测试
   1. 系统测试环境

测试环境如下表6-1，表6-2所示：

表6-1服务器环境

| **类型** |  | **配置参数** |
| --- | --- | --- |
| 硬件 | 处理器  内存 | Intel(R) Platinum 8255C  4GB |
|  | 硬盘 | 60GB |
|  | 带宽 | 6Mbps |

表6-2软件环境

| **类型** |  | **配置参数** |
| --- | --- | --- |
| 软件 | 编程语言  编程工具 | Python、Java  Pycharm、IDEA |
|  | 系统 | Ubuntu |
|  | 其他工具 | Docker |

* 1. 数据库API服务功能测试

通过Postman工具对数据库API进行测试，并在数据库中查看数据是否正确更新，测试结果如表6-3所示：

表6-3 数据库API功能测试表

| **测试项目** | **测试描述** | **测试结果** |
| --- | --- | --- |
| 判断商品是否入库  存储商品信息 | 以GET方式发送需要判断的商品链接，返回布尔值  以POST方式发送JSON格式数据，返回站点名称 | 成功  成功 |
| 获取商品信息 | 以GET方式发送商品链接，返回JSON结构商品信息 | 成功 |
| 存储站点指纹 | 以POST方式发送站点链接和指纹信息，返回布尔值 | 成功 |
| 获取站点指纹 | 以GET方式请求，返回所有站点JSON格式指纹信息 | 成功 |
| 站点初始化 | 以GET方式发送站点链接，返回布尔值 | 成功 |

* 1. 数据展示后端功能测试

前后端对接完成后，通过在前端页面操作，来测试数据展示后端各个模块运行是否正常，测试结果如表6-4所示：

表6-4数据展示后端功能测试表

| **测试项目** | **测试描述** | **测试结果** |
| --- | --- | --- |
| 态势总览功能 | 查看节点监控情况、数据分类是否正常 | 成功 |
| 交易监控功能 | 查看词云、饼状图等图表是否正常显示 | 成功 |
| 节点监控功能 | 查看根据关键词是否能够搜索出相应节点 | 成功 |
| 溯源分析功能 | 查看是否能够根据站点链接，显示站点信息表格 | 成功 |
| 搜索引擎功能 | 查看是否能够根据关键词显示搜索结果 | 成功 |
| 在线接口文档 | 查看文档是否正常显示，在线调试功能是否正常 | 成功 |

1. 总结与展望
   1. 总结

本文以综合运用了《计算机网络》、《数据库原理及应用》、《面向对象程序设计》等课程知识，通过查阅资料、需求分析等步骤确定了研究的目标和内容，并结合实际确定了本文的研究思路，采用Python、Java等语言开发一个暗网信息监测分析系统。具备目标站点发现、站点信息采集、暗网搜索引擎、数据可视化展示等功能，解决了有关部门对于暗网网络空间测绘的实际需求。本文中监测系统的设计综合运用了数据采集、爬虫调度、数据库操作、微服务开发等技术，降低了暗网信息监测的门槛，同时提供高效情报发现、关联分析和挖掘服务，为相关部门准确获取网络犯罪线索，加速破获网络犯罪提供了简单有效的解决方案。

* 1. 展望

本文针对暗网难以监控、不易追踪这一问题提供了一个有效的解决方案，为有关部门打击利用暗网违法犯罪提供了一个行之有效的工具，但仍存在着一些有待加以改进的地方。首先由于Tor代理的不稳定性，有时会存在系统接入Tor网络耗时较长或断开的问题，后续可以针对这一问题着重优化链路建立过程加快Tor的连接速度。其次，可以引入深度学习算法，在采集网站数据时进行更精确的文本特征提取与筛选。最后，为了能对系统运行中出现的错误进行及时解决，可以引入监控系统，对运行过程中的错误通过邮件或短信形式对开发人员进行告警，提高系统的健壮性。

参考文献

[1]汤艳君,安俊霖.暗网案件的爬虫取证技术研究[J].中国刑警学院学报,2018(05):115-118.DOI:10.14060/j.issn.2095-7939.2018.05.020.

[2] 于浩佳,陈波,刘蓉.匿名网站信息爬取技术研究[J].信息安全研究,2017,3(10):922-931.

[3] Cafarella M J, Madhavan J, Halevy A.Web-scale Extraction of Structured Data[J].Acm Sigmod Record, 2009, 37 (04) :55-61.

[4] Nunes E, Diab A, Gunn A, et al.Darknet and Deepnet Mining for Proactive Cybersecurity Threat Intelligence[C].IEEE Conference on Intelligence and Security Informatics, 2016:7-12.

[5] 黄莉峥,刘嘉勇,郑荣锋,李孟铭.一种基于暗网的威胁情报主动获取框架[J].信息安全研究,2020,6(02):131-138.

[6] 王建攀,金涛.暗网取证的现有困境及解决措施[J].广东公安科技,2021,29(03):8-10.

[7] 黄紫斐,刘洪梅,张舒.基于暗网环境的网络恐怖主义及其治理[J].信息安全与通信保密,2018(12):50-62.

[8] Simon Woodhead. Monitoring bad traffic with darknets[J]. Network Security,2012,2012(1).

[9] ElBahrawy Abeer,Alessandretti Laura,Rusnac Leonid,Goldsmith Daniel,Teytelboym Alexander,Baronchelli Andrea. Collective dynamics of dark web marketplaces.[J]. Scientific reports,2020,10(1).

[10] 杨溢.基于 Tor 的暗网空间资源探测技术研究[D].上海交通大学,2018.

[11] 张冬.东南亚暗网犯罪态势及国际执法合作[J].现代世界警察,2022(02):46-50.

[12] 马金强,田俊静,苗志宏.我国暗网犯罪分析与综合治理研究[J].武警学院学报,2021,37(12):27-32.

[13] 宋胜男. 暗网域名收集与内容分析方法研究[D].北京交通大学,2019.

[14] Antonio Montieri,Domenico Ciuonzo,Giuseppe Aceto,Antonio Pescape. Anonymity Services Tor, I2P, JonDonym: Classifying in the Dark (Web)[J]. IEEE Transactions on Dependable and Secure Computing,2020,17(3).

[15] Basheer Randa,Alkhatib Bassel,Xu Zhiyong. Threats from the Dark: A Review over Dark Web Investigation Research for Cyber Threat Intelligence[J]. Journal of Computer Networks and Communications,2021,2021.

[16] 苏再添,张莹莹,黄志炜.暗网取证研究[J].信息安全与通信保密,2021(03):99-107..

[17] 张伟伟,王万.暗网恐怖主义犯罪研究[J].中国人民公安大学学报(社会科学版),2016,32(04):13-18.

[18] Lan Jinghong,Liu Xudong,Li Bo,Li Yanan,Geng Tongtong. DarknetSec: A novel self-attentive deep learning method for darknet traffic classification and application identification[J]. Computers & Security,2022(prepublish).

[19] 王腾飞,蔡满春,岳婷,芦天亮.Histogram-XGBoost的Tor匿名流量识别[J].计算机工程与应用,2021,57(14):110-115.

[20]郑献春,王瑞,闫皓楠,赵兴文,李晖,李凤华.基于分布式爬虫的高性能Tor网络内容监控系统[J].信息安全学报,2023,8(01):144-153.DOI:10.19363/J.cnki.cn10-1380/tn.2023.01.11.