

## 1 文献综述

### 1.1 选题背景与应用价值

在科技持续发展进步的今天，计算机的普及越来越广泛。同时随着网络技术的迅速发展，大量多样化的网络资产为人们的生产、生活提供了极大便利，同时也对其自身的安全管理提出了挑战。准确、全面地进行网络资产探测是实现网络资产有效管理的前提，也是进行威胁分析的基础<sup>[1]</sup>。另外，网络资产不但可能存在安全风险，还很容易导致社会资源的浪费，比如电力、人力。一个庞大的网络资产体系中并非所有的网络资产都是有价值的，其中也存在无用的网络资产依旧在耗费电力和人力。网络资产探测是指追踪、掌握网络资产情况的过程，通常包括主机发现、操作系统识别、服务识别等，是实现网络安全管理的重要前提，在网络安全相关工作中具有广泛的应用价值<sup>[1]</sup>。网络资产探测对识别在线的存活资产也有极大的帮助，可以发现空转的系统，协助网络整改，避免资源的浪费，这也是响应国家节能减排整体战略的一项重要举措，具有十分重要的意义和研究价值<sup>[2]</sup>。

因此，为了降低甚至避免网络资产中存在的安全威胁以及协助网络资产整改工作，进行网络资产扫描工作就十分有必要。网络资产扫描可以高效地发现一个网络资产体系中存活的所有网络资产，并在一定程度上检测可能存在的漏洞风险，进而方便专业人士排除网络资产系统中存在的安全问题，以达到保护资产安全的目的<sup>[10]</sup>。另外，网络资产的扫描结果对网络资产整改也具有十分深远的意义，可以极大地提高资产利用率。

本系统是基于 ZMap 设计和实现的，旨在实现一个即可以提供网络资产扫描又可以进行主机资产监控和扫描，且方便用户操作和使用，同时可以进行用户权限管理的资产扫描系统。本系统在网络资产扫描方面考虑到扫描速度的限制，因此支持批量扫描，即下发一个扫描任务之后无需等待扫描结果，同时可以进行其他的一些操作。在主机资产扫描方面考虑到外部扫描的局限性，因此支持下载插桩程序，进行主机资产的内部监控和扫描。本系统从两个不同的层面进行资产扫描，即可以对外进行网络资产的扫描，也可以对内进行主机资产的监控和扫描，具有综合性，并且尽可能地追求数据的完整性和准确性，具有很强的实际应用价值。

### 1.2 国内外研究现状分析

首先对网络资产扫描工具的研究现状分析。近年来，在网络扫描技术这个领域涌现过众多优秀的工具和先进的理念。其中就有大家最为熟悉的网络扫描工具 Nmap，Nmap 是基于响应协议栈指纹的网络资产探测工具的典型代表，该工具通过向目标资产发送请求连接数据包或构造的其他各层协议数据包，根据响应数据包的特征进行主机发现和端

口、服务、操作系统等的识别<sup>[3]</sup>。时至今日，仍然还有许多企业以 Nmap 为底层扫描核心来制作大型的网络资产扫描应用<sup>[4]</sup>。本系统也在一些需要深入扫描的业务场景中借鉴了 Nmap 的部分功能来满足业务需求。但是 Nmap 也有一个很明显的问题，那就是扫描速度不是很理想，常常需要等待较长的时间。另外还有一种更加快速的扫描工具，那就是 ZMap，也是本系统核心最底层所采用的扫描工具。Zmap 基于异步无状态扫描工具对扫描机制进行了改进，但通常只能进行端口扫描和主机发现，对操作系统、服务及应用的探测则无能为力，其以弱化功能全面性为代价来增强探测时效性<sup>[5]</sup>。本系统则充分合并了两者的特点，同时在确保数据可靠的情况下尽可能地提高扫描速率。

然后对网络资产扫描系统的研究现状进行分析。早在 2015 年，Durumeric 等将 ZMap 同谷歌云平台相结合，开发了 Censys 系统，该系统使用了基于 Go 语言开发的 ZGrab 应用扫描器与 Zmap 配合，而后进行数据处理汇总，提取结构化数据，并将其保存于谷歌云存储平台；利用开源的 ElasticSearch 平台和谷歌 BigQuery 分别在前端和后台为用户提供 Zmap 全网端口、服务扫描结果的搜索查询<sup>[6]</sup>。这种方式固然有效，但是，所获取的数据并不实时，只能提供之前的数据用于后续分析。另外，该系统也不具备对内网的资产扫描功能，其主要核心功能还是围绕着外网 IP 段来进行展开的。早在 2009 年，Matherly 发布了一款网络设备扫描引擎，其可以扫描所有与互联网关联的设备，Shodan 通过扫描全网设备并抓取解析各个设备返回的 banner 信息，通过了解这些信息 Shodan 就能得知设备相关的所有信息并进行分析<sup>[7]</sup>。

### 1.3 研究内容与研究条件

本网络资产扫描系统在 B/S 结构的基础上，又将服务端以微服务<sup>[8]</sup>的形式进行拆分，这样即方便管理也方便横向扩展。本系统总体包括网页端和服务端，服务端可拆分为网页接口端和业务逻辑端。开发工具统一使用 Vim 进行开发。网页接口端使用 Golang 语言进行开发，并且其底层所使用的 Web 框架是 IRIS 框架，最终会生成一个可执行文件，然后以 Docker 多层构建的方式部署到阿里云服务器上运行。

系统的业务逻辑端采用 Golang 语言以及 Shell 脚本进行开发，其包含多个小型的服务，这些服务之间以 RPC 和 Kafka 消息队列的方式进行数据的交换和通信，最终会生成多个可执行文件，然后统一使用 Linux Supervisor 工具统一进行部署和管理。另外，考虑到微服务的本质特征，采用 Etcd 来进行服务的注册与发现，这样可以比较高效地动态管理微服务之间的联系，而不需要手动时刻关注。系统缓存使用的是 Redis。数据库使用的是 MongoDB。消息队列使用的是 Kafka。为了方便缓存、数据库、消息队列和服务注册与发现的部署和使用，Kafka、Redis、MongoDB 和 Etcd 都采用容器的方式进行部署。因此在技术上是可以实现的。

开发此网络资产扫描系统所需的物理投资为笔记本一台和阿里云服务器两台以及若干测试机，在经济上是可行的。

本系统操作简单、界面简洁，同时所有服务均是一次部署随处可用，所需要耗费的资源较小，并且用户在使用时并不需要耗费任何资源，可以直接在网页端进行操作，因此在操作上是可行的。

## 1.4 论文结构

正文部分主要分为七个章节，每个部分的主要内容如下：

第一章为文献综述，对选题的背景和应用价值进行了讲述，然后对国内外研究现状进行了分析，最后介绍了本系统的研究内容与研究条件。

第二章为开发使用到的部分技术的介绍，主要介绍了 ZMap、gRPC、Kafka、Etcd、Docker、Redis 等技术，这些技术是网络资产扫描系统设计和实现的基础。

第三章为需求分析，对系统整体进行了概述，然后分别从功能需求和非功能需求两个方面进行了分析，确定了开发此系统大体需要完成的工作。

第四章为系统设计，从系统整体架构设计和系统功能设计两个方面介绍了系统的总体设计，介绍了各个模块的主要功能，然后介绍了系统四个主要的功能模块的设计和系统数据库的设计，对数据库中的集合进行了详细说明。

第五章为系统实现，首先搭建系统的基本框架，然后结合图片和具体代码对四个模块的主要界面、接口实现和服务实现进行了详细的介绍，最后介绍了阿里云服务器部署的主要步骤。

第六章为系统测试，介绍了测试理论、测试环境和四个模块的测试内容和测试用例，然后对测试结果进行分析。

第七章为结论，对系统背景意义以及系统的优点和存在的问题进行了简要的总结。

## 2 相关技术介绍

### 2.1 ZMap

ZMap 是 Github 上开源的网络扫描工具，因其扫描速度快而被广泛应用。ZMap 使用了无连接的扫描技术，每次探测都没有进行完整的三次 TCP 握手，因此可以显著提高扫描速度。ZMap 的核心功能是进行主机发现和开放端口检测。ZMap 以命令行形式为应用的基础，故其可以很轻松地嵌入到一个大型应用中并做为网络资产快速扫描的核心组件进行应用和扩展。

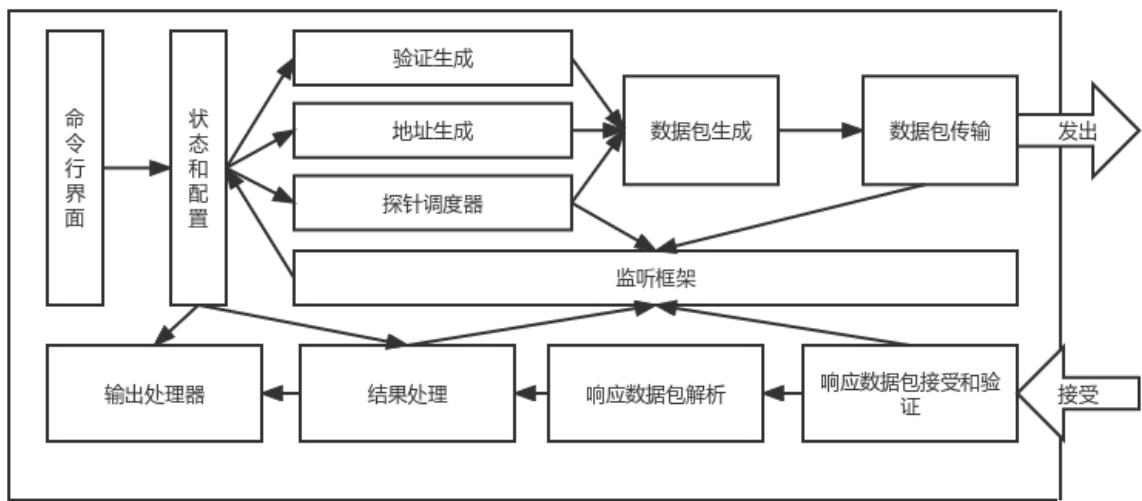


图 2.1 ZMap 网络扫描器的系统结构

首先分析 ZMap 网络扫描器的系统结构。ZMap 根据功能将系统拆分成多个模块，各个模块之间相互通信，这增加了系统的灵活性和扩展性，使得 ZMap 可以轻松地集成其他的工具和应用，以便构造功能强大的网络资产扫描系统。如图 2.1 所示，ZMap 扫描器的核心部分由状态、配置解析、验证模块、目标 IP 地址生成模块、监听模块和数据包发送、接收、处理模块共同组成。另外还有一个探针调度器，ZMap 会根据不同的网络协议使用不同的探针模块去验证目标设备的存活。ZMap 的目标 IP 地址生成模块采用了 IP 地址生成分片技术，利于迅速生成探测数据包。由于 ZMap 在进行网络扫描时不会建立 TCP 连接，其所能获取到的信息就会比较少，但是其在针对多台设备某一端口的开放扫描时扫描速度却很快。本系统则充分利用了 ZMap 对端口快速扫描的优势并将其作为最底层的扫描核心组件加以使用。

## 2. 2 gRPC

**gRPC** 是一个现代的开源高性能 RPC 框架，可以在任何环境中运行。它可以有效地连接数据中心内和跨数据中心的服务，支持负载均衡、跟踪、健康检查和身份验证。它也适用于分布式计算中，将设备、移动应用程序和浏览器连接到后端服务。**gRPC** 使用 **Protocol Buffers** 定义服务，自动以各种语言和平台生成通用的客户端和服务器存根，可以提高开发效率<sup>[20]</sup>。**gRPC** 支持用一行代码安装运行时和开发环境，十分简洁方便，并且使用这个框架可以将 RPC 访问扩展到每秒数百万次。**gRPC** 底层是基于 **HTTP/2** 进行通信的，并且还支持双向流的数据传输，效率较高<sup>[9]</sup>。总之，**gRPC** 框架极大地降低了服务端开发和运维的复杂度。

## 2. 3 Kafka

**Kafka** 是一个分布式发布-订阅消息系统，已成为了一个很好的大规模消息处理应用的解决方案。其本质就是发布者将消息发送到某个数据通道，然后消费着从这个数据通道中取出数据并进行相应的处理。**Kafka** 不仅支持分布式集群部署，大大提高吞吐效率和系统扩展性，其接受到的消息数据还是以文件数据的形式持久化存储的，直到消息数据过期。这样做可以确保消息数据不会因为系统错误而丢失，容错性能高<sup>[14]</sup>。另外，因为上述特性，**Kafka** 还支持多个消费者同时消费同一数据通道中的数据，十分符合本系统的业务需求。

## 2. 4 Etcd

**Etcd** 是一个高度一致的分布式键值存储系统。它提供了一种可靠的方式来存储需要由分布式系统或集群访问的数据，其在 **Kubernetes** 集群中应用十分广泛，常常做为服务注册与服务发现的代表应用<sup>[12]</sup>。由于本系统服务端采用的是微服务的设计架构，为了方便两两服务之间可以相互发现并进行通信，故而采用了单节点 **Etcd** 来进行服务的注册与发现。这样做方便不仅方便系统的扩展，而且不需要手动操作，可以极大提高服务端微服务的开发和维护效率。

## 2. 5 Docker

**Docker** 作为云原生的发起源，在近几年应用十分广泛。**Docker** 可以十分轻松地创建一个系统应用，而不需要用户知道底层细节，操作十分简便。**Docker** 创建的应用本质上还是一个进程，只是由于 **Docker** 采用了 **Linux** 系统所提供的 **Namespaces** 和 **CGroups** 特性，可以将 **Docker** 所创建的进程进行隔离和资源限制。另外，**Docker** 采用分层镜像的理念，结合写时复制的设计思想，使高效创建应用成为可能<sup>[13]</sup>。本系统为了

高效地部署一些应用，也使用了 Docker 进行应用部署。这样做可以极大地提高系统部署效率而且不会污染计算机系统的文件系统，一举两得。

## 2.6 Redis

Redis 是一个键值存储系统，是跨平台的非关系型数据库。它基于内存运行，性能十分高效。并且具有十分丰富的数据类型和高并发性<sup>[18]</sup>。上述特征表明，Redis 十分适用于数据的缓存。另外，Redis 也支持订阅和发布的消息队列，但由于存在数据可能丢失以及允许发送的消息不能过大等限制，本系统只采用 Redis 来进行数据的缓存。总之，这样做可以极大地提高关键数据的响应速率，加快系统运行。

## 2.7 本章小结

本章第一节介绍了 ZMap 相关技术，主要介绍了 ZMap 的系统结构和优缺点，这些是网络资产扫描系统业务逻辑端开发需要的最基础的技术，同时也是扫描组件的核心部分；第二节介绍了 gRPC 框架，这个框架是服务端服务之间进行通信的核心；第三节和第四节分别介绍了 Kafka 和 Etcd，这两个应用是业务逻辑端服务之间进行数据传输和服务发现的基础。第五节介绍了 Docker 技术，这个技术是实现应用部署以及应用扩展的核心技术。第六节介绍了 Redis 应用，该数据库是实现数据快速响应以及后续 JWT 验证的基础<sup>[17]</sup>。以上的技术和框架是网络资产扫描系统实现的基础，这些技术和框架之间相互联系、相互合作，共同完成系统所提供的功能。

### 3 需求分析

#### 3.1 项目概述

该网络资产扫描系统的核心内容是基于 GoLang 语言进行开发的，数据展示网页是基于 TypeScript 语言进行开发的，采用的是分布式微服务架构进行设计和部署，旨在设计并实现一个易于开发维护部署<sup>[19]</sup>、功能完善、方便用户使用、界面整洁的资产扫描系统。另外，该系统还实现了用户登录、主机资产扫描、网络资产扫描、用户管理等一系列的功能。同时，根据用户需求分析设计出来的用例图如图 3.1 所示。

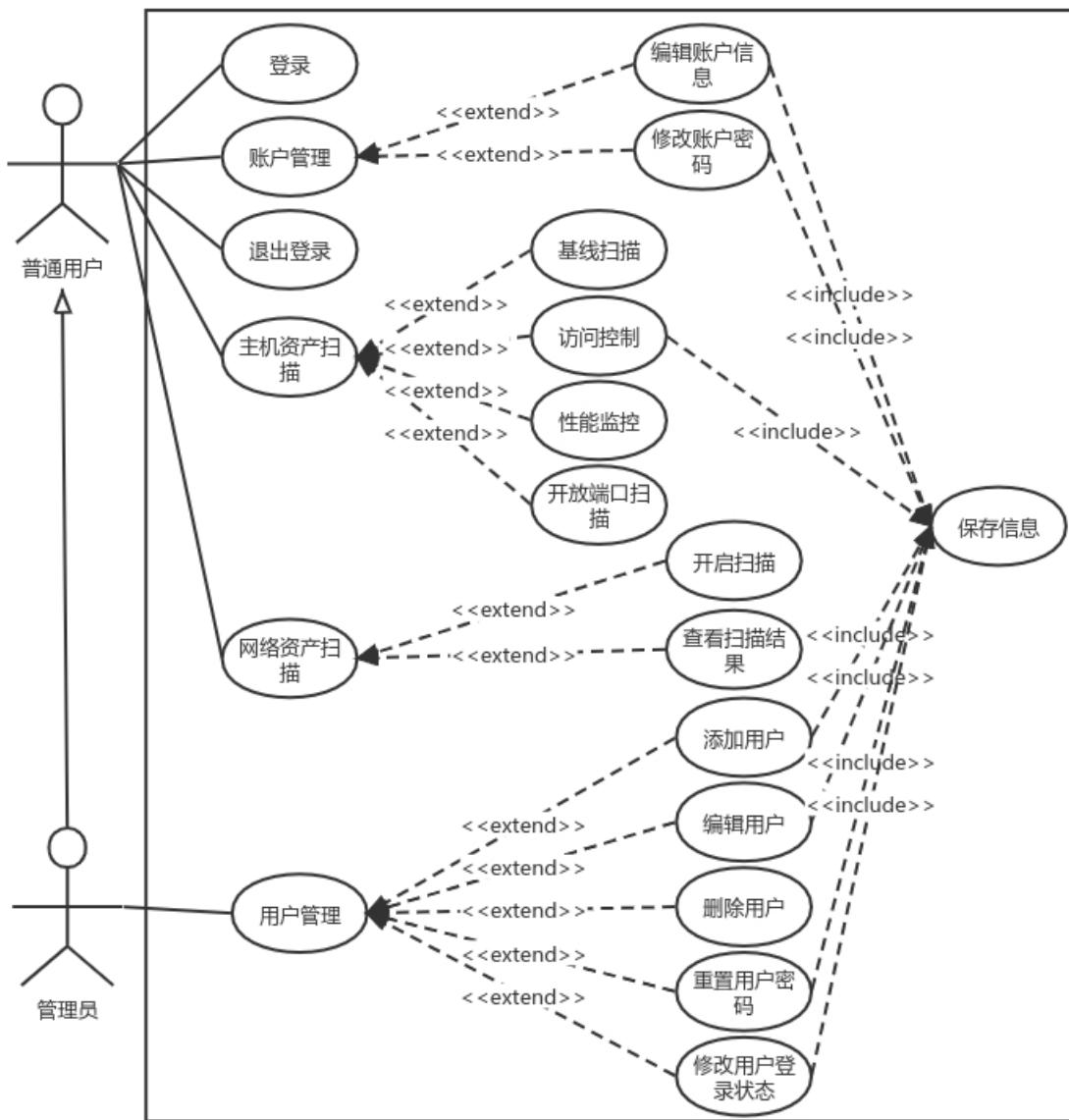


图 3.1 网络资产扫描系统用例图

### 3.2 功能需求

该网络资产扫描系统主要分为登录注销、主机资产扫描、网络资产扫描和用户管理四个模块。登录注销模块提供了登录界面和注销按钮，提供了用户登录和注销功能，是后续所有功能的基础。主机资产扫描模块又包含基线扫描界面、访问控制界面、性能监控界面和开放端口扫描界面，主要功能是监控和检测主机资产的系统信息和安全状态。在网络资产扫描模块中，又包含网络资产扫描界面和网络资产扫描结果界面这两个核心功能。本系统在用户层面进行了权限控制，即只有拥有相应权限的用户才可以访问相应的界面和接口。用户管理模块针对用户权限进行了子模块的划分，分别为账户管理界面和用户管理界面。当用户以普通权限进行登录时，其只能操作账户管理模块，即编辑当前用户信息和修改当前用户密码。当用户以管理员权限进行登录时，其还可以操作用户管理模块，即添加新用户、编辑用户信息、修改用户的登录状态和重置用户密码。因此，本系统并未在其他层面提供用户注册功能。接下来分模块来详细介绍功能需求。

(1) 登录注销模块的需求分析如表 3.1 所示，分别对两个界面中的各个功能和组件进行了详细介绍。

表 3.1 登录注销模块需求

序号	大分类	小分类	详细说明
1	登录界面	用户名输入框	登录时用户可以在输入框输入用户名
		密码输入框	登录时用户可以在输入框输入密码
		验证码输入框	登录时用户可以在这个输入框内输入验证码
		图片验证码	用户在访问登录界面时后台自动生成的图片验证码
2	注销界面	登录按钮	点击后可以将用户输入的用户名、密码、验证码和图片验证码 ID 提交给网页接口端进行验证
		注销按钮	点击后弹出确认框
		确认框	确认框中提示用户是否确认退出登录
		确认按钮	点击后退出登录，返回登录界面
		取消按钮	点击后取消注销操作

(2) 主机资产扫描模块的需求分析如表 3.2 所示，分别对各个功能和组件进行了详细介绍。

表 3.2 主机资产扫描模块需求

序号	大分类	小分类	详细说明
1	主机资产界面	批量删除按钮	未选择主机资产项时，该按钮置灰。选择了多个主机资产项时，该按钮可点击，点击后可批量删除主机资产项
		搜索框	可以根据输入的主机资产 IP 地址来搜索对应的主机资产
2	性能监控界面	主机资产列表	显示监控的主机资产，并且可以点击删除按钮删除当前项或者点击详情按钮进入性能监控界面
		主磁盘监控表单	动态显示当前主机主磁盘的关键信息
		资源监控表单	显示当前主机的 CPU 核数以及内存总量
		CPU 使用率图	以折线图的形式动态显示 CPU 使用率
3	访问控制界面	内存使用率图	以折线图的形式动态显示内存使用率
		安全组规则列表	使用本系统创建的规则会显示在该列表中，且可以点击删除按钮删除当前项
		安全组规则作用方向下拉框	选择点击作用方向来过滤安全组规则列表
		新建规则按钮	点击后弹出填写安全组规则的信息框，信息框右下方有确定按钮和取消按钮
4	开放端口扫描界面	确定按钮	点击后发送请求给网页接口端，创建成功后规则会显示在安全组规则列表中
		取消按钮	点击后关闭信息框，取消新建规则操作
		端口类型下拉框	选择点击端口类型过滤开放端口列表
		端口状态下拉框	选择点击端口类型过滤开放端口列表
5	基线扫描界面	搜索框	可以根据输入的端口号、端口服务或 PID 来搜索对应的端口列表
		开放端口列表	动态显示当前主机资产的开放端口信息，且可以点击详情按钮查看详情
		扫描结果下拉框	选择点击扫描结果过滤扫描结果列表
		扫描按钮	点击后可以立刻执行基线扫描操作
6	分页界面	扫描结果列表	显示当前主机资产基线扫描后的结果，针对不通过的项可以点击建议按钮查看解决方案或点击忽略按钮忽略
		扫描结果详情表单	汇总扫描结果的所有信息，方便用户第一时间排查问题
		列表总项数	凡是涉及到列表的地方都显示项总数
		分页按钮	点击后跳转到指定页中显示列表内容
		分页输入框	输入页数后跳转到指定页

(3) 网络资产扫描模块的需求分析如表 3.3 所示, 分别对各个功能和组件进行了详细介绍。

表 3.3 网络资产扫描模块需求

序号	大分类	小分类	详细说明
1	网络资产界面	批量删除按钮	未选择网络资产扫描项时, 该按钮置灰。处于正在扫描状态中的扫描项无法被删除, 选择了一个或多个网络资产扫描项时, 该按钮可点击, 点击后可批量删除网络资产扫描项
		搜索框	可以根据输入的扫描项 CIDR 来模糊搜索对应的网络资产扫描项
		新建扫描列表	点击后弹出一个表单, 输入指定的内容后点击确定可以发送网络资产扫描的请求, 同时会将扫描项显示在网络资产扫描列表中
2	网络资产扫描详情界面	网络资产扫描列表	分页显示当前用户创建的扫描项, 并且可以点击删除按钮删除当前项或点击详情按钮进入网络资产扫描详情界面
		扫描结果汇总表	显示扫描成功率、扫描类型、扫描选项、扫描耗时等信息
		端口分布图	以柱形图或折线图的形式综合显示扫描结果的端口分布情况
3	IP 扫描详情界面	服务分布图	以柱形图或折线图的形式综合显示扫描结果的服务分布情况
		已扫描 IP 列表	显示扫描成功的所有的 IP 地址, 点击 IP 地址进入 IP 扫描详情界面
		端口服务标签	默认选择。点击后会显示探测启动提示和端口服务列表
		探测启动提示	提示是否开启服务探测和脚本探测
		端口服务列表	显示端口服务最基本的信息, 点击服务详情会弹出更加详细的探测信息。当未开启服务探测和脚本探测选项时, 该按钮置灰
		路由追踪标签	当开启路由追踪选项时该标签可点击。点击后以路径图的形式显示经过的路由信息, 同时会显示路由追踪所依据的端口号和协议类型
		系统探测标签	当开启系统探测选项时该标签可点击。点击后以扇形图的形式显示目标 IP 操作系统的类型和可能性

(4) 用户管理模块根据用户权限进行了划分, 与普通用户权限相关的账户管理模块需求分析如表 3.4 所示, 与管理员权限相关的用户管理模块需求分析如表 3.5 所示。两表分别对各个功能和组件进行了详细介绍。

表 3.4 账户管理模块需求

序号	大分类	小分类	详细说明
1	账户管理界面	账户信息表单	显示了当前账户的相关信息, 比如用户角色, 用户名, 密码强度等信息
		编辑账户图标	点击后可以直接在表单修改当前账户的信息, 点击确认按钮后发送请求给网页接口端。
		修改账户密码图标	点击后弹出修改密码框, 输入原密码以及两次新密码后点击确认按钮进行密码修改, 点击取消按钮则取消操作

表 3.5 用户管理模块需求

序号	大分类	小分类	详细说明
1	用户管理界面	用户列表	以分页列表的形式展示当前系统的所有用户信息
		登录状态下拉框	选择点击登录状态过滤用户列表
		用户角色下拉框	选择点击用户角色过滤用户列表
		添加用户按钮	点击后弹出侧边框, 输入新用户信息后点击确认按钮发送请求给网页接口端, 点击取消按钮则取消操作
		批量删除按钮	未复选框选择用户时置灰, 否则点击后可以删除被选择的所有用户
		搜索框	输入用户名, 邮箱或联系方式可以搜索用户列表中符合条件的用户并显示
		登录状态开关	点击可修改相应用户的登录状态, 开为允许登录, 关为禁止登录
		编辑图标	点击后可以修改指定用户的账户信息
		删除图标	点击后可以删除指定用户
		重置图标	点击后弹出重置密码框, 输入重置密码后点击确认可以重置指定用户的登录密码

### 3.3 非功能需求

#### (1) 网页界面需求

网页的界面是用户与系统进行交互的媒介，是最直接的，网页界面的质量直接关系到用户对整个系统的使用体验。本系统在网页的排版以及呈现形式上都符合现代网页的简约设计风格，每个系统功能块都以导航的形式提供，各个界面之间的配色也比较同一，十分醒目。数据的展示方式也一目了然，方便用户的使用。

### (2) 易用性需求

本系统操作简单，所有的复杂功能都在后台运行，用户只需要点击鼠标就可以实现比较复杂的功能，另外，系统展示的内容都以十分通俗的文字来进行表示，语义十分明确。系统方便了用户的使用与操作，极大地提升了用户的使用体验。

### (3) 安全性需求

本系统中的所有数据都需要在登录后才可以访问，除了获取验证码和登录外，其他的所有请求都需要使用 **JWT token**，只有验证通过后才会执行相应的操作。系统的服务器使用的是阿里云服务器，其本身就具有防止网络攻击的功能，因此可以较好地保护数据的安全。为了避免网页数据在传输过程中被他人截获，系统采用了 **HTTPS** 来保护用户与网页之间的连接。另外，为了避免用户登录后长时间未操作所存在的风险，系统设置了超时未操作自动退出登录功能。

## 3.4 本章小结

本章第一节进行了项目概述，第二节分四个模块对系统的功能需求进行了详细说明，最后一节从网页界面需求、易用性需求和安全性需求三个方面对系统的非功能需求进行了概述。

## 4 系统设计

### 4.1 系统总体设计

基于 ZMap 的网络资产扫描系统整体采用了 B/S 架构进行开发, 即网页端和服务端。服务端又可分为网页接口端和业务逻辑端。业务逻辑端又采用了分布式微服务架构进行组织与设计。同时在服务端还会对数据库进行管理和操作。网页端主要功能是发送请求给网页接口端并接受和处理响应以及展示从网页接口端获取到的数据。网页接口端的主要功能是响应网页发送过来的请求, 然后经过一系列比较简单的逻辑处理后将响应的数据打包发送给网页端。业务逻辑端则是整个系统的核心, 处理一些比较耗时, 比较重要的业务逻辑。在业务逻辑端, 根据功能的不同将服务进行拆分, 服务之间通过 RPC 或 Kafka 消息队列的方式进行通信或数据的传输, 这些服务之间相互合作, 共同为系统的核心业务提供支持。

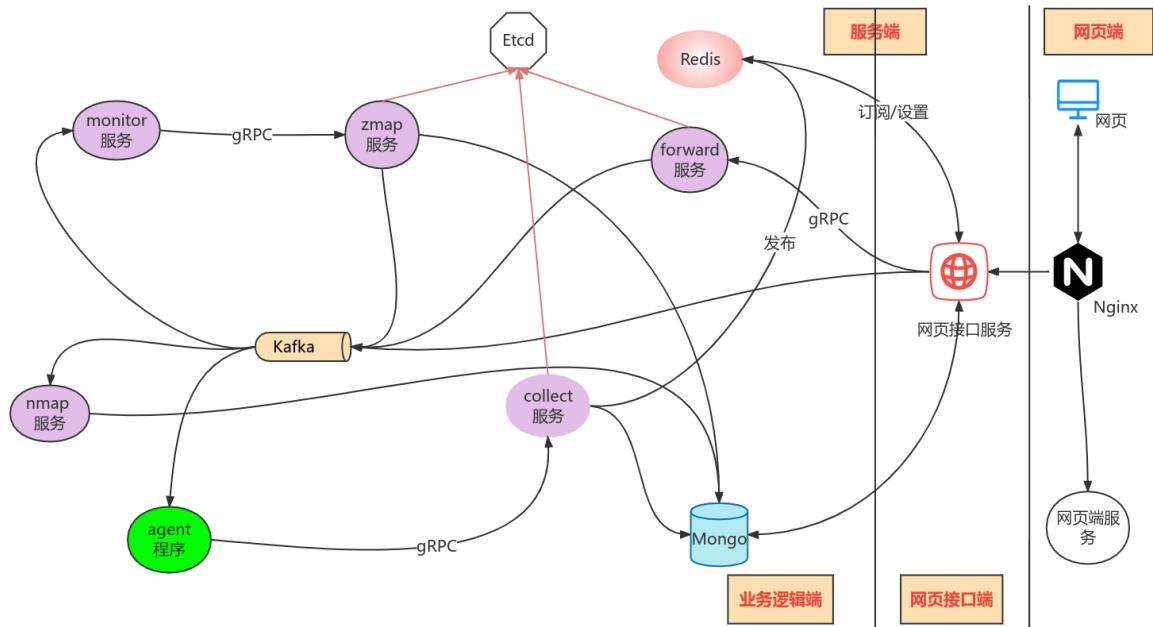


图 4.1 网络资产扫描系统架构图

网络资产扫描系统的总体架构如图 4.1 所示。

在网页端, 网页端服务用于表示网页的内容以及网页的界面组成, 其最终的表示形式就是网页。Nginx 提供网关和反向代理的功能的<sup>[16]</sup>。当在浏览器输入网页地址后,

Nginx 会反向代理到网页端服务，进而展示网页的内容。网页发送的请求又会通过 Nginx 反向代理到网页接口端的网页接口服务。

网页接口端的网页接口服务用于响应网页的请求同时返回请求的数据。它会利用 Redis 来进行数据的缓存以及同步，另外，在必要的情况下会将一些核心的业务逻辑转发给业务逻辑端进行同步/异步处理，最终，数据都会存储到 MongoDB 数据库中，这样做的好处就是网页端可以快速地收到响应数据，提升用户体验。

在业务逻辑端，一共包括五个微服务，分别是 forward 服务、monitor 服务、zmap 服务、nmap 服务和 collect 服务。forward 服务与网页接口服务采用 gRPC 进行通信，用于转发网页接口服务所提供的网络资产扫描数据，然后将获取到的数据初步处理之后发布到 Kafka 消息队列中，其主要功能就是提供异步高并发功能。monitor 服务会订阅 Kafka 消息队列，取出相应的数据之后进行整合过滤然后以 gRPC 的方式发送给 zmap 服务。zmap 服务接受到扫描数据之后会并发执行初步的扫描任务，扫描结束之后会将初步的扫描结果存入 MongoDB 数据库中，同时更新扫描状态，然后在必要时会将一些关键数据发布到 Kafka 消息队列中。nmap 服务则用来提供深入扫描功能，其会订阅 Kafka 消息队列然后取出数据进行扫描，扫描结束之后会将扫描结果存入到 MongoDB 数据库中，同时更新扫描状态。至此，一套完整的网络资产扫描就完成了。collect 服务用于收集主机资产的信息，经过相应的处理之后存入 MongoDB 数据库中，另外，collect 服务在收到主机资产的基线扫描结果之后会同步 Redis 缓存。

在业务逻辑端还提供了一个 agent 程序，该程序是用于用户下载执行的代理程序，用户可以将该程序安装到任意主机上，之后就可以在网页中控制或获取主机资产的行为与数据，该程序就是执行主机资产扫描的核心程序。

Etcd 则用于 zmap 服务、forward 服务和 collect 服务进行服务注册与发现。这样做的好处就是可以自动进行服务发现，不用进行硬配置，可以极大提高系统的灵活性以及适应性，也十分方便对服务进行管理和系统扩展。

#### 4.1.2 系统功能结构

基于 ZMap 的网络资产扫描系统主要分为登录注销、主机资产扫描、网络资产扫描和用户管理四个模块，功能模块图如图 4.2 所示：

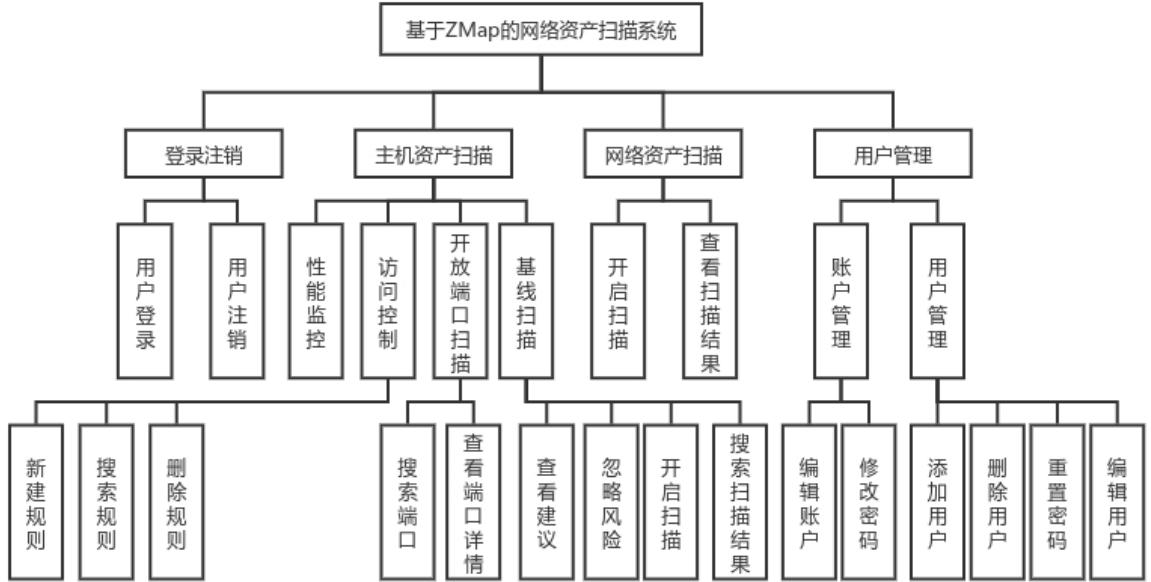


图 4.2 网络资产扫描系统功能模块图

每个模块的功能如下：

- (1) 登录注销模块：主要分为用户登录和用户注销两部分。管理员或者管理员创建的允许登录的用户可以进行登录，登录后的用户才可以进行注销。
- (2) 主机资产扫描模块：主要包括性能监控、访问控制、开放端口扫描和基线扫描四部分。性能监控动态展示注册了的主机资产的关键系统资源信息。访问控制用于限制主机资产的网络访问，包括新建规则、搜索规则和删除规则功能，其功能类似于阿里云服务器的安全组控制。开放端口扫描会实时更新主机资产的开放端口详情，包括搜索开放端口和查看开放端口信息功能，为用户提供网络服务信息。基线扫描用于检测主机资产的配置是否合规，包括开启扫描、搜索扫描结果、查看修复建议和忽视风险项等功能。
- (3) 网络资产扫描模块：主要包括新建网络资产扫描和查看扫描结果两部分。在新建网络资产扫描时，可以根据需要选择不同的扫描类型和扫描可选项，其中，扫描类型包括快速扫描、普遍扫描、端口范围扫描、端口扫描和端口服务扫描，可选项包括是否开启脚本探测功能、是否开启系统探测功能、是否开启服务探测功能、是否开启路由探测功能。在扫描结束之后，则可以查看扫描结果，扫描结果分为两部分，一部分是综合展示，一部分是选择查看扫描 IP 的扫描结果。
- (4) 用户管理模块：主要包括账户管理和用户管理两部分。普通用户只能操作账户管理，即查看当前账户信息、编辑账户信息和修改账户密码。管理员用户则可以额外

操作用户管理，即添加新用户、编辑已存在的用户信息、删除已存在的用户、重置用户密码和修改已存在用户的登录状态。

## 4.2 系统详细设计

### 4.2.1 系统网页请求处理设计

本系统是以网页发起请求并接受响应的形式来进行数据的获取的。为了满足安全性需求和性能需求，需要对网页请求进行安全性校验，同时又可以通过校验信息获取用户相关的数据且不会耗费系统资源。

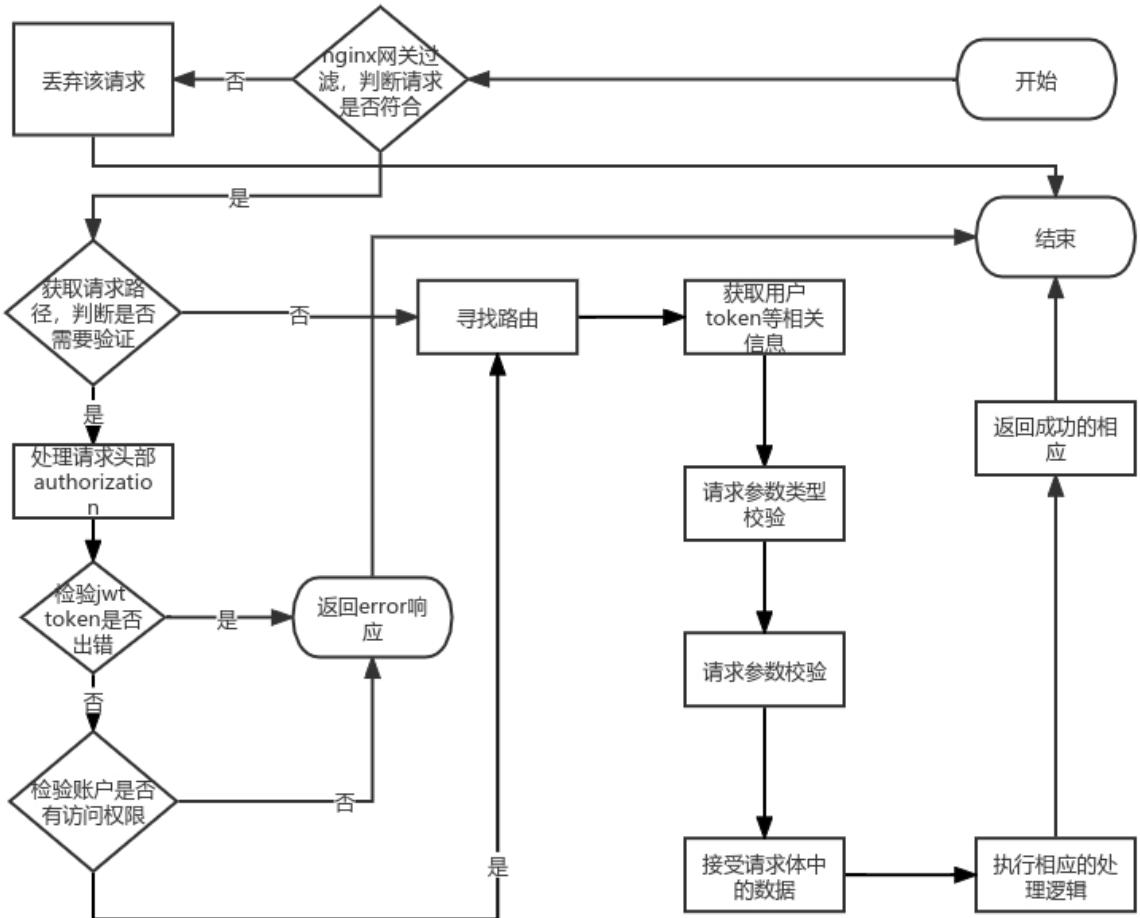


图 4.3 网页请求处理逻辑流程图

当网页发起请求时，请求会首先经过 Nginx 网关，如果请求不符合规则，则将该请求丢弃，反之，请求会通过 Nginx 的反向代理进入网页接口端进行处理。在网页接口端首先对请求路径进行判断，如果请求路径不需要进行 token 验证，则直接执行寻找路由

的操作，反之，网页接口端会获取请求头部的 `authorization` 字段的内容，该字段保存了用户登录时获取的 JWT token 信息。然后对 `token` 进行有效性校验，如果 `token` 过期、不存在白名单中、已加入黑名单中或出错，则返回 `Unauthorized` 响应，反之，再检验账户是否具有访问请求路径的权限，如果没有，则返回权限不足错误，反之，则继续执行寻找路由操作。然后再获取 `token` 中的用户信息进行保存，方便后续用户数据获取。然后再对请求的参数类型和请求参数进行安全性校验，这样做可以避免用户在网页请求上进行注入攻击。接下来就是接受请求中的数据，然后执行相应的处理逻辑，执行完毕之后返回成功的响应，如图 4.3 所示。

#### 4.2.2 系统功能模块设计

##### (1) 登录注销模块设计：

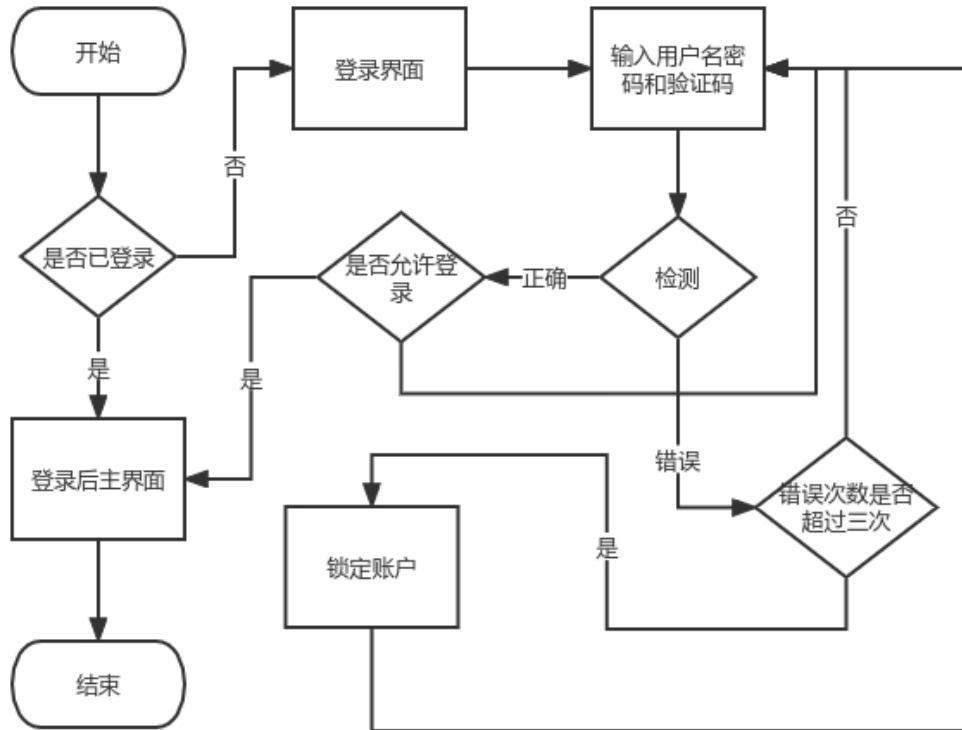


图 4.4 登录注销模块流程图

当用户进入网络资产扫描系统时，若用户没有登录或已注销，会进入登录界面，若用户已登录，则会进入登录后的主界面。在登录界面中，用户需要输入用户名、用户密码和验证码，全部填写完毕之后可以点击登录按钮，网页端在确认用户所填信息不为空后会把表单信息以 POST 请求的方式发送给网页接口端，如果验证成功则会返回用户 ID、

用户名、JWT 和是否允许用户登录的标志等信息，之后网页端就会跳转到登录后的主界面，如果验证失败，网页端接收到验证失败的请求之后会弹出相应的错误提示信息。如果用户登录错误三次以上，用户账户会锁定 5 分钟，在此期间禁止登录此账户。如果用户账户被管理员锁定为禁止登录或用户无账户，这时用户需要询问管理员获取账户登录权限或创建新账户。用户登录之后，点击注销按钮或 JWT 过期之后会跳转到登录界面。如图 4.4 所示。与之对应的登录界面的原型图如图 4.5 所示。

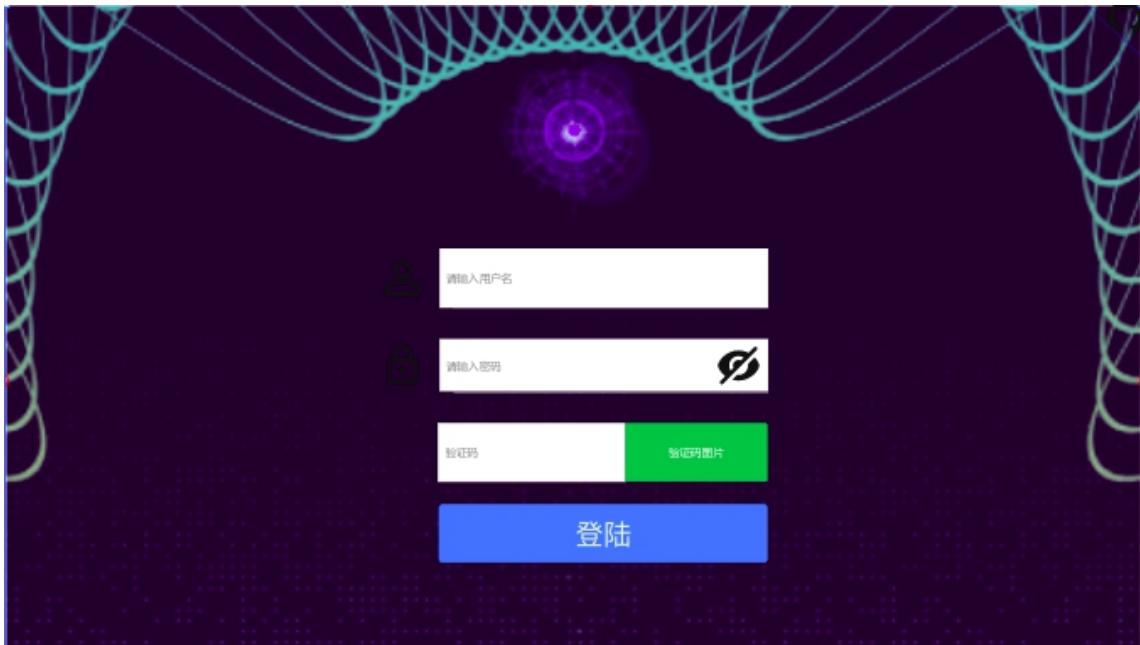


图 4.5 登录界面原型图

## (2) 主机资产扫描模块设计：

当用户登录成功之后，会自动跳转到主机资产界面。该界面中会展示被监控的主机资产列表，列表项包括主机地址、主机名、运行状态、安装时间、代理程序 PID、更新时间和操作。若此列表为空，用户可以在帮助中心下载最新版的代理程序并在需要被监控的主机上运行即可，系统会自动识别并显示对应的主机资产信息。用户可以根据搜索框中输入的内容对主机地址进行模糊搜索，另外，当主机状态是离线时，可以选择点击批量删除按钮或操作中的删除按钮删除主机监控，当主机状态是运行时，可以点击详情按钮对主机资产执行进一步的操作。为了提升网页端的响应速度，主机列表的获取采用分页的方式进行。上述所有行为在网页端都是通过发送相应的请求给网页接口端来对数据进行操作。主机资产扫描原型图如图 4.6 所示。

## 基于 ZMap 的网络资产扫描系统设计与实现

主机地址	主机名	状态	安装时间	Agent Pid	更新时间	操作
127.0.0.1	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.5	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.2	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.6	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.10	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.9	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.8	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.7	aico	运行中	2014-12-24 23:12:00	1123	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.3	aico	离线	2014-12-24 23:12:00	-	-	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.4	aico	离线	2014-12-24 23:12:00	-	-	<a href="#">删除</a>   <a href="#">详情</a>

图 4.6 主机资产扫描原型图

当用户点击详情按钮后，会进入主机资产详情界面。其中包括性能监控、开放端口扫描、访问控制和基线扫描四个部分。在性能监控界面中，网页端间歇性地向网页接口端获取主机资产数据，进而动态显示主机资产的的主磁盘信息、系统资源信息、CPU 使用率和内存使用率。如图 4.7 所示。



图 4.7 性能监控界面原型图

## 基于 ZMap 的网络资产扫描系统设计与实现

在开放端口扫描界面中，以分页的形式展示主机资产对外开放端口的列表。其列表项包括端口号、端口类型、端口状态、端口服务、PID 和操作。用户可以点击操作中的详情查看更加详细的端口信息，也可以使用搜索框、端口类型选择和端口状态选择来过滤列表信息。如图 4.8 所示。

端口号	端口类型	端口状态	端口服务	PID	操作
1000	tcp	LISTENING	abcdef	1	详情
1000	tcp6	LISTENING	abcdef	2	详情
1000	udp	-	abcdef	3	详情
1000	udp6	-	abcdef	4	详情

图 4.8 开放端口扫描界面原型图

在访问控制界面中，用户可以对主机资产的网络访问进行控制，点击新建规则按钮后会弹出一个表单，选择或填入相应的信息之后网页端会检测数据是否合规，检测通过则发送新建规则请求给网页接口端，新建成功的规则会显示在规则列表中。规则列表项包括作用方向、授权策略、协议类型、作用端口、授权对象、描述、创建时间和操作。用户也可以点击操作中的删除按钮删除不符合的规则。如图 4.9 所示。

The screenshot shows the Access Control interface. At the top, there are tabs: 性能监控 (Performance Monitoring), 开放端口扫描 (Open Port Scan), 访问控制 (Access Control) (which is selected), and 基线扫描 (Baseline Scan). Below the tabs is a search bar and two dropdown menus for '端口类型' (Port Type) and '端口状态' (Port Status), both set to '全部' (All). A large blue button labeled '新建规则' (Create Rule) is visible. The main area displays a table of rules with columns: 作用方向 (Action Direction), 授权策略 (Authorization Strategy), 协议类型 (Protocol Type), 作用端口 (Port), 授权对象 (Authorization Object), 描述 (Description), 创建时间 (Creation Time), and 操作 (Operation). The table lists several rules, each with a '删除' (Delete) button in the '操作' column. A modal dialog titled '新建规则' (Create Rule) is overlaid on the table. It contains fields for '授权策略' (Authorization Strategy) (set to '允许' (Allow)), '授权对象' (Authorization Object) (set to '0.0.0.0/0'), '作用方向' (Action Direction) (set to '入方向' (Inbound)), '作用端口' (Port) (set to '请输入端口号' (Enter port number)), and '协议类型' (Protocol Type) (set to 'TCP'). At the bottom of the dialog are '确认' (Confirm) and '取消' (Cancel) buttons.

图 4.9 访问控制界面原型图

在基线扫描界面中，用户可以点击扫描按钮对主机资产进行基线扫描。扫描结果会以表单和扫描项列表的形式展示。表单中包括扫描开始时间、扫描结束时间、检查项数、合规项数和违规项数。扫描项列表以分页的形式进行展示，其中包括扫描项名称、扫描结果和操作。当扫描结果违规时，用户可以点击操作中的建议按钮查看修复方式、点击忽略按钮忽略此条违规项或点击恢复按钮恢复忽略项，另外，通过扫描结果下拉框可以过滤相应的扫描项。如图 4.10 所示。

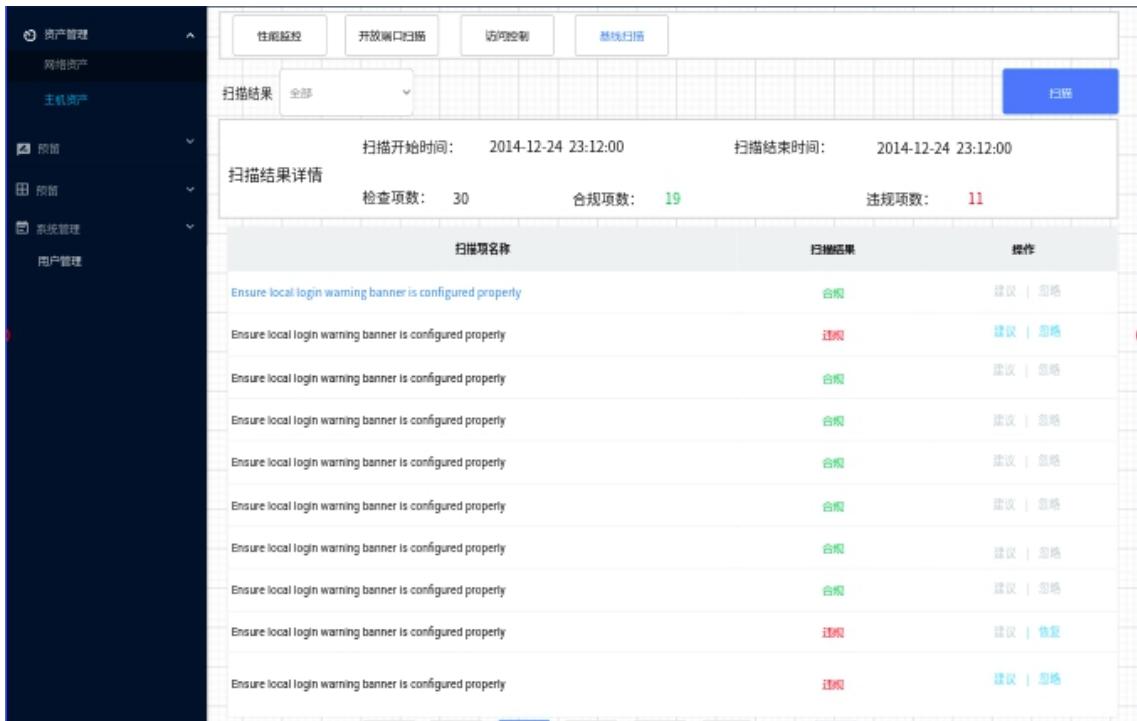
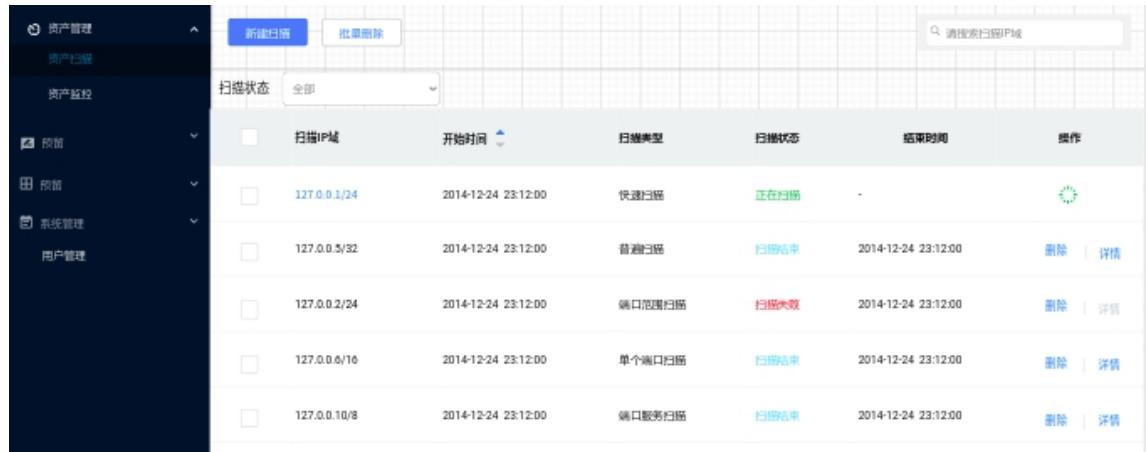


图 4.10 基线扫描界面原型图

### (3) 网络资产扫描模块设计：

用户点击左侧导航栏的网络资产按钮可以跳转到网络资产扫描界面。该界面中会以分页的形式展示创建的扫描项。用户点击新建扫描按钮之后会弹出一个表单，其中包括扫描域、扫描类型、是否开启脚本探测、是否开启服务探测、是否开启路由追踪、是否开启系统探测选择，填写完毕之后点击确定，检测通过之后会将扫描请求发送给网页接口端，网页接口端将扫描任务分发给业务逻辑端之后立刻返回。此时网页端会显示新建扫描项，其中包括扫描 IP 域、开始时间、扫描类型、扫描状态、结束时间和操作。当扫描成功结束之后操作中的删除按钮和详情按钮会显示并可点击。用户可以点击删除按钮删除扫描项或点击详情按钮查看扫描结果。如图 4.11 所示。

点击详情按钮之后会进入扫描结果界面，其中包括扫描成功率、扫描用时、扫描选项等信息。另外，以柱状图或折线图的形式总结并显示扫描域中所有 IP 的端口分布图和服务分布图。同时会提供扫描域中扫描成功的所有的 IP 地址。点击 IP 地址可以查看更加详细的扫描结果，其中包括端口服务详情、路由追踪路径图和操作系统概率图等信息。



The screenshot shows a list of network scan tasks. The columns include: 扫描IP域 (Scan IP Range), 开始时间 (Start Time), 扫描类型 (Scan Type), 扫描状态 (Scan Status), 结束时间 (End Time), and 操作 (Operations). The tasks listed are:

扫描IP域	开始时间	扫描类型	扫描状态	结束时间	操作
127.0.0.1/24	2014-12-24 23:12:00	快速扫描	正在扫描	-	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.5/32	2014-12-24 23:12:00	普通扫描	扫描结束	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.2/24	2014-12-24 23:12:00	端口范围扫描	扫描失败	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.6/16	2014-12-24 23:12:00	单个端口扫描	扫描结束	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>
127.0.0.10/8	2014-12-24 23:12:00	端口服务扫描	扫描结束	2014-12-24 23:12:00	<a href="#">删除</a>   <a href="#">详情</a>

图 4.11 网络资产扫描界面原型图

#### (4) 用户管理模块设计：



The screenshot shows the user information management interface. It includes tabs for 个人信息 (Personal Information) and 用户管理 (User Management). The current view is under 个人信息. The form fields include:

- 账号信息 (Account Information)
- 用户角色: 管理员 (User Role: Administrator)
- 用户名: admin (Username: admin)
- 注册邮箱: 2237616014@qq.com (Email: 2237616014@qq.com)
- 联系手机: 18743256457 (Mobile: 18743256457)
- 密码强度: 低 (Password Strength: Low) - indicated by a red bar
- 创建时间: 2014-12-24 23:12:00 (Creation Time: 2014-12-24 23:12:00)

At the bottom are two buttons: 确认 (Confirm) and 取消 (Cancel).

图 4.12 账户信息界面原型图

如图 4.12 所示。用户点击导航栏的账户信息后，会跳装到账户信息界面。其中包括账户角色、账户 ID、账户名、账户邮箱、联系方式、创建时间和密码强度。另外，用户可以点击相应的编辑图标编辑账户信息和修改账户密码。



图 4.13 用户管理界面原型图

如图 4.13 所示。为方便进行用户权限的管理。专门为管理员账户提供一个用户管理界面。用户列表中包含所有的用户信息，其中包括用户名、用户角色、登录状态、用户邮箱、用户联系方式、创建时间和操作。管理员可以通过点击添加用户按钮或批量删除按钮增加或删除用户，也可以点击登录状态开关控制用户是否允许登录。操作中的图标从左到右依次对应编辑用户信息、删除用户和重置密码。为方便管理员对用户进行筛查，提供了登录状态、用户角色和搜索框进行用户过滤。

#### 4.2.3 系统数据库设计

(1) 本系统数据库有十个集合：用户集合、安全组规则集合、网络资产扫描状态集合、基线扫描结果集合、基线扫描规则集合、基线扫描项集合、网络资产预扫描结果集合、网络资产深扫描结果集合、主机资产集合和主机资产详情集合。

##### (2) 数据库集合设计

① 用户集合 `tb_user` 用来存储用户的基本信息。如表 4.1 所示。有九个字段，分别是用户 ID、用户名、用户类型、是否允许登录、密码、密码强度、邮箱、联系方式和上次登录时间。

表 4.1 `tb_user` 集合

列名	数据类型	注释	约束
uid	Int32	用户编号	主键约束
username	String	用户名	唯一性约束
user_type	Int32	用户类型	
enable	Bool	是否允许登录	
password	String	MD5 加密密码	
password_strength	Int32	密码强度	

② 安全组规则集合 **tb\_secgrp** 用来存放用户在指定主机上新增的安全组规则，包括主机资产 ID、规则 ID、作用方向、协议类型、应用策略、作用端口、作用域、创建时间等。如表 4.2 所示。

表 4.2 tb\_secgrp 集合

列名	数据类型	注释	约束
_id	ObjectId	文档编号	主键约束
agent_id	String	主机资产编号	
rule_id	String	规则编号	唯一性约束
direction	Int32	作用方向	
protocol_type	Int32	协议类型	
apply_policy	Int32	应用策略	
port	Int32	作用端口	
cidr	String	作用域	
create_time	Int64	创建时间	

③ 网络资产扫描状态集合 **tb\_scan\_status\_desc** 存储了扫描项的基本信息，包括用户 ID、开始时间、扫描域、扫描类型、预扫描是否结束、预扫描结束时间、深扫描是否结束、深扫描结束时间、扫描状态等。如表 4.3 所示。

表 4.3 tb\_scan\_status\_desc 集合

列名	数据类型	注释	约束
_id	String	文档编号	主键约束、唯一性约束
uid	Int32	用户编号	
start_time	Date	开始时间	
scan_ip	String	扫描域	
scan_type	String	扫描类型	
is_first_scan_done	Bool	预扫是否结束	
first_scan_done_time	Date	预扫结束时间	
is_deep_scan_done	Bool	深扫是否结束	
deep_scan_done_time	Date	深扫结束时间	
status	Int32	扫描状态	

④ 基线扫描结果集合 `tb_scan_cis_results` 存储了主机资产执行基线扫描后的扫描结果，包括分组编号、主机资产 ID、基线规则 ID、扫描结果、是否忽略等。如表 4.4 所示。

表 4.4 `tb_scan_cis_results` 集合

列名	数据类型	注释	约束
<code>_id</code>	<code>ObjectId</code>	文档编号	主键约束
<code>id</code>	<code>Int32</code>	分组编号	
<code>agent_id</code>	<code>String</code>	主机资产编号	
<code>cis_id</code>	<code>String</code>	基线规则编号	
<code>status</code>	<code>Bool</code>	扫描结果	
<code>is_ignored</code>	<code>Bool</code>	是否忽略	

⑤ 基线扫描规则集合 `tb_repo_cis` 存储了以 Linux Fedora 操作系统为例的基线规则，包括基线规则 ID、规则名、描述、基本原理、修复方案等。如表 4.5 所示。

表 4.5 `tb_repo_cis` 集合

列名	数据类型	注释	约束
<code>_id</code>	<code>ObjectId</code>	文档编号	主键约束
<code>id</code>	<code>String</code>	规则编号	唯一性约束
<code>name</code>	<code>String</code>	规则名称	
<code>description</code>	<code>String</code>	描述	
<code>rationale</code>	<code>String</code>	基本原理	
<code>remediation</code>	<code>String</code>	修复方案	

⑥ 基线扫描项集合 `tb_scan_cis` 存储了主机资产基线扫描的结果汇总，包括主机资产 ID、开始时间、结束时间、扫描总项数、合规项数、分组编号等。如表 4.6 所示。

表 4.6 `tb_scan_cis` 集合

列名	数据类型	注释	约束
<code>_id</code>	<code>ObjectId</code>	文档编号	主键约束
<code>agent_id</code>	<code>String</code>	主机资产编号	
<code>start_time</code>	<code>Int64</code>	开始时间	
<code>end_time</code>	<code>Int64</code>	结束时间	

count	Int32	扫描总项数	
success_count	Int32	合规项数	
id	Int32	分组编号	唯一性约束

⑦ 网络资产预扫描结果集合 **tb\_first\_scan\_result** 存储了针对指定扫描域发现的存活主机的 IP 和扫描选项等信息。如表 4.7 所示。

表 4.7 tb\_first\_scan\_result 集合

列名	数据类型	注释	约束
_id	String	文档编号	主键约束、唯一性约束
start_time	Date	开始时间	
ip_domain	String	扫描域	
end_time	Date	结束时间	
count	Int32	存活主机数	
result	String Array	主机 IP 数组	
deep_scan_option	Object	深扫描选项	

⑧ 网络资产深扫描结果集合 **tb\_deep\_scan\_result** 存储了最终扫描的结果详情，包括扫描域、耗费时间、开始时间、结束时间、扫描数、主机详情等。如表 4.8 所示。

表 4.8 tb\_deep\_scan\_result 集合

列名	数据类型	注释	约束
_id	String	文档编号	主键约束、唯一性约束
start_time	Date	开始时间	
deep_scan_ip	String	扫描域	
end_time	Date	结束时间	
elapsed	Double	扫描消耗时间	
hosts	Object Array	主机扫描详情	
count	Int32	扫描数	

⑨ 主机资产集合 **tb\_agent** 存储了注册到系统的主机资产的基本信息，包括主机资产 ID、主机资产 IP、主机名、代理程序进程号、主机 MAC 地址、加入时间、最近上线时间、是否被删除等。如表 4.9 所示。

表 4.9 tb\_agent 集合

列名	数据类型	注释	约束
_id	ObjectId	文档编号	主键约束
hash_id	String	主机资产编号	唯一性约束
agent_ip	String	主机资产 IP	
hostname	String	主机名	
pid	Int64	进程号	
mac_address	String	主机 MAC 地址	
join_time	Int64	加入时间	
update_time	Int64	最近上线时间	
is_deleted	Bool	软删除标记	

- ⑩ 主机资产详情集合 tb\_agent\_info 存储了主机资产的系统详情等信息。如表 4.10 所示。

表 4.10 tb\_agent\_info 集合

列名	数据类型	注释	约束
_id	ObjectId	文档编号	主键约束
hash_id	String	主机资产编号	唯一性约束
disk_info	Object	主磁盘信息	
resources	Object	系统资源信息	
port_infos	Object Array	端口服务信息	

### 4.3 本章小结

本章的第一节详细讲述了系统的整体架构设计和系统模块的结构设计，对各个功能模块的详细结构进行了剖析；第二节介绍了系统详细设计，首先介绍了系统网页请求的处理逻辑，然后对四个主要的功能模块的原型展示进行了设计，最后对数据库中的十个集合文档进行了详细说明。

## 5 系统实现

### 5.1 系统基本框架搭建

本系统分为网页端、网页接口端和业务逻辑端三个部分，分别对应三个目录 `nas-ui`、`nas-web` 和 `nas-daemon`。由于 `nas-web` 和 `nas-daemon` 具有结构体共用、数据库接口共用和 RPC 接口共用的情况，把两者的公共部分剥离成一个新目录 `nas-common`。RPC 接口的生成需要编写对应的 `proto` 文件，为方便统一管理，单独创建一个新目录 `protos` 用于存放所有的 `proto` 文件，并编写对应的 Shell 脚本使自动生成的接口代码会被移动到 `nas-common` 目录下的指定位置，方便 `nas-web` 和 `nas-daemon` 进行调用。另外，为了简化 MongoDB、Redis、Kafka、Etcd、Gitea 和 Jenkins 的部署方式，使用 `docker-compose.yaml` 配置文件进行容器化部署。接下来，将依次针对各个部分进行详细的介绍。

#### 5.1.1 网页端框架搭建

网页端使用 TypeScript 语言进行开发，结合使用了 Ant Design 框架和 UmiJS 框架进行设计。首先进入 `nas-ui` 目录，执行 `yarn create @umijs/umi-app && yarn` 命令进行初始化，另外，为了方便对网页端的代码进行版本控制和集成部署<sup>[15]</sup>，还要编写一个 `Jenkinsfile` 文件，然后使用 `git remote add` 命令将代码仓库引入，在每次进行推送前需要使用 `yarn build` 命令生成网页静态文件。

#### 5.1.2 网页接口端框架搭建

网页接口端使用 Golang 语言进行开发，结合使用了 Iris Web 框架、JWT 等技术。首先是目录结构的搭建，进入 `nas-web` 目录，分别创建 `cmd`、`config`、`confile`、`dao`、`controller`、`deployment`、`internal`、`pkg`、`middleware`、`router`、`service`、`support` 和 `webutils` 目录，`cmd` 目录用于存放程序的入口，`config` 目录用于存放配置文件反序列化后的结构体声明和解析函数，`confile` 目录用于存放配置文件，`dao` 目录用于存放请求或响应数据的反序列化结构体、MongoDB 和 Redis 的函数封装，`controller` 目录用于存放相应路由请求的执行函数，`deployment` 目录用于存放项目部署的配置文件，`internal` 目录用于存放 Redis 连接函数、MongoDB 连接函数、Kafka 连接函数、Iris 包装器的封装函数等一些项目私有的功能函数，`pkg` 目录用于存放 kafka 使用函数的封装和 RPC 接口调用的函数封装，`middleware` 目录用于存放请求处理的中间件，`router` 目录用于存放路由相关的函数封装，`service` 目录用于存放请求最终执行的函数定义，`support` 目录用于存放请求响应函数和一些常量的声明，`webutils` 目录用于存放工具函数。然后使用

go mod init nas-web 命令创建 go.mod 文件，该文件用于管理引入包的信息。为了使 nas-common 中的内容可以在 nas-web 中引用，需要在 go.mod 文件中增加 replace nas-common => ../nas-common 内容。

接下来是系统框架搭建，进入 internal 目录，新建目录 wrapper，进入 wrapper 目录，新建 mcontext.go 文件，Iris 框架使用的是 [github.com/kataras/iris/v12 v12.1.8](https://github.com/kataras/iris/v12)，在其中定义 Context 结构体，包括 iris.Context 和 models.UserToken 两部分内容，iris.Context 是 Iris 框架自身提供的与请求处理相关的接口，models.UserToken 是定义在 nas-common 中的结构体，其中包括两部分信息：用户 ID 和用户类型，这么做的好处是当解析请求时就可以将存储在 JWT 中与用户相关的信息提前存储到该结构体对象中，方便之后的请求执行函数进行调用，然后定义了一个 Handler 函数，该函数形参类型是 func(\*Context)，与之对应的就是 controller 目录下的请求执行函数，返回值类型是 iris.Handler，该函数内容就是返回一个参数为 iris.Context 类型，无返回值的函数。最终，所有的请求都会调用 Handler 函数进行 JWT 中用户数据的获取、请求的执行和资源的释放。另外，为了进行请求参数类型校验和请求参数校验，新建 wrapper.go 文件，在其中定义函数 ApiWrapper(ctx \*Context, handler func(ctx \*Context, reqBody interface{}) error, paramChecker bool, reqBody interface{}, params ...interface{})，该函数会通过传入的 paramChecker 来判断是否开启参数校验和 params 来判断请求参数的类型，如果出错，会调用 support 目录中定义的 SendApiErrorResponse 函数返回出错响应，否则继续调用 checkParam 函数进行参数校验，checkParam 中又会调用 validate.Struct 第三方库函数进行校验，若校验通过，会调用 handler(ctx,reqBody) 进行请求的真正执行。接下来是初始化路由操作，在 router 目录下新建 router.go 文件，在其中定义 func InitRouters(app \*iris.Application) 函数，通过调用 app.Party 函数进行路由分组操作，然后再通过定义相应的注册函数 func(router.Party) 就可以将一系列路由都注册到相应的路由分组上。

网页接口端采用的是 JWT 鉴权方式，将 JWT 以中间件的形式插入到请求处理前的逻辑中来进行鉴权。JWT 使用的是 [github.com/dgrijalva/jwt-go v3.2.0+incompatible](https://github.com/dgrijalva/jwt-go v3.2.0+incompatible)，进入 middleware 目录，新建并进入 jwt 目录，新建 config.go 文件，在其中新建一个 Config 结构，其中主要包括上下文键、错误处理函数、JWT 提取函数、JWT 密钥解析函数、加密方式等。另外，新建 jwt.go 文件，在其中定义全局变量 jwt \*struct{Config Config}、lock 互斥锁、JWT 密钥和 SysToken 用于防止 JWT 重放。再定义一个 ConfigJWT 函数，函数体中以加锁方式执行 jwt 全局变量的配置，包括加密方式 jwt.SigningM256、错误处理函数、JWT 提取函数 GetAuthHeader、JWT 密钥获取函数等，然后再定义一个参数为 context.Context 的 CheckJWT 函数，函数体中首先调用 jwt 全局变量中配置的

JWT 提取函数进行 JWT 提取并保存到 `token` 变量中，如果 `token` 为空则返回错误消息，否则继续检测 `token` 是否在黑名单中，若在黑名单中则表示用户已经退出登录，否则继续检测 `token` 是否在白名单中，若在白名单中则对存储在 Redis 中的 JWT 进行续期，之后则调用 jwt 全局变量中配置的 JWT 密钥获取函数对 `token` 进行解析，之后就是进行数据加密算法和数据字段的校验，上述校验都通过之后会将 JWT 中存储的用户信息存储到 `models.UserToken` 结构体变量中，同时将变量设置为上下文的值，之后的请求处理就可以从上下文中获取到用户信息并存储到自定义的上下文中。进入 `middleware` 目录并新建 `iddleware.go` 文件，在这里面定义一个 JWT 中间件函数 `JwtMiddleware`，函数体中主要执行对请求路径的检测，若请求路径是配置文件中配置的不需要鉴权的路径，则跳过 JWT 鉴权，否则需要对请求进行拦截同时进行 JWT 校验。

网页接口端的整体框架十分灵活，若需要新增路由接口，只需要在路由中新增一条路由，然后在 `controller` 目录和 `service` 目录中新增一条请求入口函数和请求执行函数即可。

### 5.1.3 业务逻辑端框架搭建

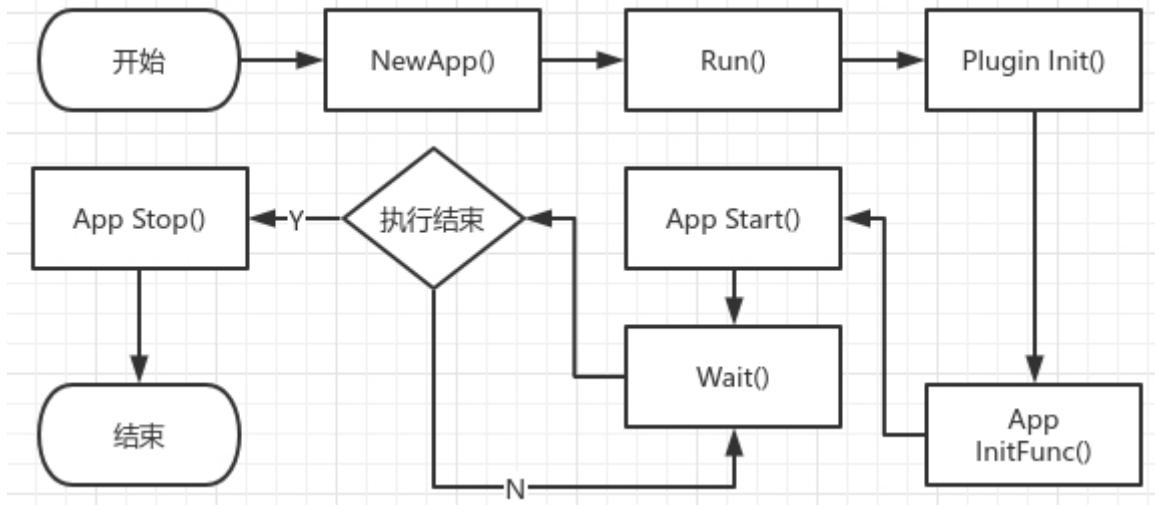


图 5.1 业务逻辑端 APP 执行流程

业务逻辑端采用的是微服务的形式进行设计开发的，为了使所有的服务具有统一的调用接口，定义了一个 `App` 结构体，该结构体包含 `InitFunc func()error`、`StartFuncList []func()error`、`StopFunc func()error`、`PluginList []Plugin`、`stopOnce sync.Once`，分别对应 `App` 初始化函数、`App` 启动函数列表、`App` 停止函数、插件列表(所有的插件必须实现 `Plugin` 接口中的 `Init()error` 函数和 `Stop()error` 函数)、同步控制。通过定义 `NewApp(...)*App` 函数来新建 `App`，另外，定义 `func(app *App)Init()error` 来进行真正

的初始化操作，`func(app *App) Stop()`来进行真正的停止操作，`func(app *App)Start()`来进行真正的启动操作。最终，通过定义 `func(app *App)Run()` 函数来作为 App 真正的运行入口，在其中会依次调用所有插件的 `Init()` 函数、`app.Init()` 函数、`app.Start()` 函数、`app.Stop()` 函数和依次调用所有插件的 `Stop()` 函数。在 `app.Start()` 函数中，通过调用 `threading` 库函数 `NewRoutineGroup()` 来执行所有的 App 启动函数，同时调用 `Wait()` 函数阻塞所有启动函数，直到全部执行完毕，如图 5.1 所示。这样做使得 MongoDB 和 Redis 都可以做为插件的形式载入需要的服务中运行，十分方便对每个服务进行开发和管理维护。

## 5.2 登录注销模块实现

登录注销模块界面主要包括一个登录界面和一个注销按钮，如图 5.2 所示。接口主要包括登录接口、验证码获取接口和注销接口，该模块不涉及到与业务逻辑端的交互。

### 5.2.1 主要界面

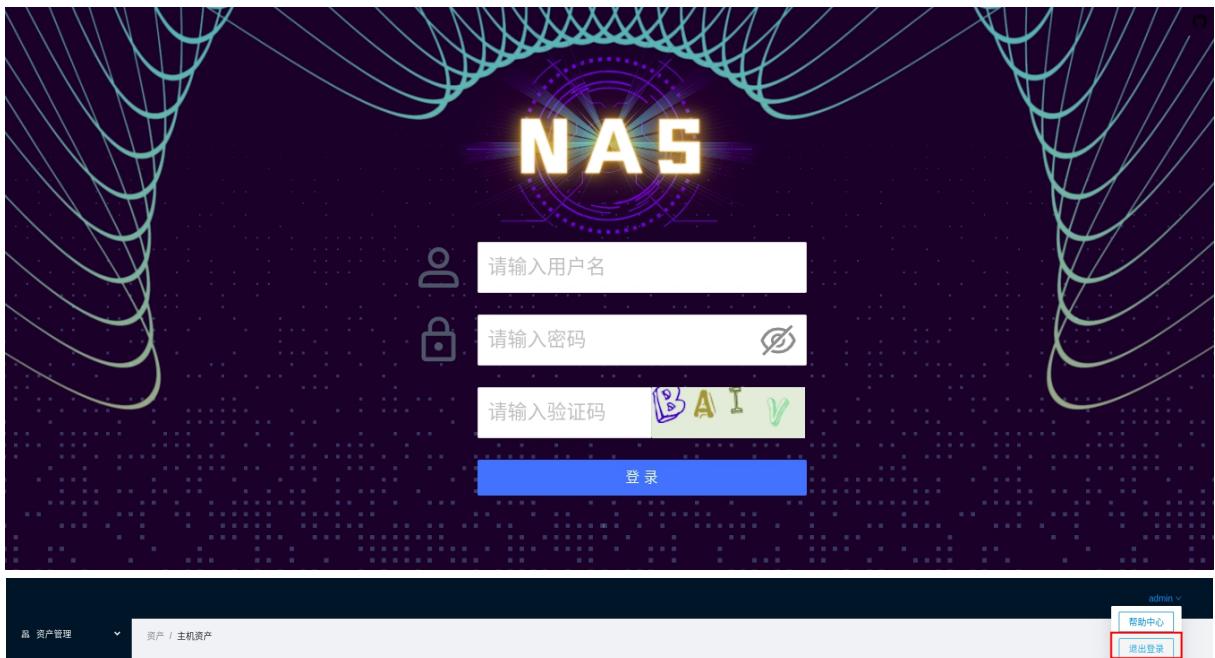


图 5.2 登录注销模块主要界面

(1) 登录界面：使用了灵活布局，居中对齐，并设置了背景颜色和图片。输入框使用了一个表单，表单项分别对应 Ant Design 中的 `Input` 元素和传统的 `img` 元素。当进入登录页面时，会触发验证码获取 `GET` 请求，当获取到验证码 `base64` 数据后传给 `img` 的 `src` 部分，即可展示验证码图片。当用户填写完表单的内容之后，即可点击登录按钮，

之后会调用表单的 `validateFields` 函数进行有效性验证，然后通过调用 `then` 函数发送登录 `POST` 请求给网页接口端，获取到响应后通过校验返回值的 `enable` 字段来判断是否允许用户登录，若允许，则调用 `localStorage.setItem` 函数将返回值的 JWT、用户 ID 和用户名存储到本地，然后跳转到登录成功后的页面，否则会弹出相应的错误提示框。

(2) 注销按钮：用户登录成功之后，可以点击用户名下拉框中的退出登录按钮，然后网页端会发送注销 `POST` 请求给网页接口端，请求的头部还会带上 JWT 信息用于鉴权，请求处理成功之后还会调用 `localStorage.removeItem` 函数溢出本地存储的 JWT、用户 ID 和用户名，然后跳转到登录页面，否则会弹出相应的错误提示框。

### 5.2.2 接口实现

登录注销模块的路由配置定义在 `nas-web/router/api/common` 目录下的 `auth.go` 文件中，请求入口函数定义在 `nas-web/controller` 目录下的 `auth.go` 文件中，请求处理函数定义在 `nas-web/service` 目录下的 `auth.go` 文件中。

(1) 验证码获取接口：验证码获取请求方式为 `GET`，请求路径为 `/auth/verifycode/`，请求入口函数为 `VerifyCode`，该接口不需要进行 JWT 校验和参数校验，请求处理函数为 `VerifyCodeHandler`。实现流程为：通过调用 `GenDigitCaptcha` 函数可以获取到验证码 base64 数据和验证码 ID，然后调用 `SendApiResponse` 函数将生成的验证码 base64 数据和验证码 ID 返回给网页端。

(2) 登录接口：登录请求方式为 `POST`，请求路径为 `/auth/login/`，请求入口函数为 `Login`，该接口不需要进行 JWT 校验但需要进行 JSON 形式的参数校验，请求处理函数为 `LoginHandler`。实现流程为：若验证码不等于万能验证码，需要调用 `VerifyCaptcha` 函数进行验证码校验，若失败则会返回错误消息，然后通过查找数据库方式对账户名进行校验，若失败则会返回错误消息，然后通过调用  `GetUserLoginLock` 函数读取 Redis 缓存对系统安全设置进行校验，若用户已登录失败次数超过三次，会封锁账户 5 分钟并返回错误消息，然后调用 `CheckPassword` 函数对用户密码进行校验，若密码错误，会调用  `SetUserLoginLock` 函数将错误信息置入 Redis 缓存并返回错误消息，否则会调用  `RemoveUserLoginLock` 函数移除错误信息，然后检测是否允许用户登录，若允许则会调用  `GenerateToken` 函数生成 JWT 并调用  `SetJwtWhiteList` 函数将 JWT 置入 Redis 缓存白名单中，否则会返回内容为禁止登录的错误消息，然后调用 `LoginResp` 结构体构建返回消息并调用  `SetAuthCookie` 和  `SendApiResponse` 将返回消息发送给网页端。

(3) 注销接口：注销请求方式为 `POST`，请求路径为 `/auth/logout/`，请求入口函数为 `Logout`，该接口需要进行 JWT 校验但不需要进行参数校验，请求处理函数为 `LogoutHandler`。实现流程为：首先调用 `GetHeader` 函数获取请求头部的 JWT 信息，然

后通过调用 `GetTokenRemainingTime` 函数计算 JWT 剩余的过期时间，然后调用 `SetJwtBlacklist` 函数将 JWT 置入 Redis 缓存黑名单并设置过期时间，最后调用 `SendApiResponse` 函数将处理成功消息返回给网网页端。

### 5.3 主机资产扫描模块实现

主机资产扫描模块界面主要包括一个主机资产界面和四个主机资产详情界面，如图 5.3 所示。接口主要包括主机列表获取接口、主机系统信息获取接口、主机对外开放端口获取接口、代理程序下载接口、主机资产删除接口、安全组规则新增接口、安全组规则删除接口、安全组规则列表获取接口、开启基线扫描接口、基线扫描结果获取接口、基线扫描属性更新接口、基线扫描详情获取接口，该模块涉及到与业务逻辑端 `collect` 服务和 `forward` 服务的交互。

#### 5.3.1 主要界面

The screenshot displays two pages of the network asset management system:

- Host Assets Page:** Shows a table of host assets with columns: 主机IP, 主机名, Agent 状态, 安装时间, Agent 进程号, 上线时间, and 操作 (Delete, Detail). One entry is shown: 192.168.27.225, fedora, 运行中 (Running), 2022/1/9 00:40:36, 1331881, 2022/5/16 13:01:05.
- Performance Monitoring Page:** Shows performance monitoring details for the host fedora. It includes sections for **主磁盘监控** (Main Disk Monitoring) and **资源监控** (Resource Monitoring).

In the **主磁盘监控** section, it shows device name: /dev/sda1, file system type: ext4, mount point: /, and other information: rw,relatime. It also shows disk size: 329.09GB and disk usage rate: 48%.

In the **资源监控** section, it shows CPU core count: 8 and memory total: 7.52GB. It includes two line charts: **CPU 使用率** (CPU Usage Rate) and **内存使用率** (Memory Usage Rate). The CPU chart shows usage fluctuating between 0 and 50%. The memory chart shows usage stable around 70%.

## 基于 ZMap 的网络资产扫描系统设计与实现

The figure consists of three vertically stacked screenshots of a network asset management system, likely built with a Vue.js front-end framework.

- Top Screenshot (Port Scanning):** Shows the "Open Port Scan" tab selected. It displays a table of open ports on the system. The columns include Port Number, Port Type, Port Status, Service, PID, and Action. Notable entries include port 22 (tcp) listening on /usr/sbin/sshd, port 8000 (tcp) listening on /usr/bin/node, and various UDP ports (5355, 43020, 67) listening on /usr/sbin/avahi-daemon.
- Middle Screenshot (Access Control):** Shows the "Access Control" tab selected. It displays a table of network access rules. The columns include Action Direction, Authorization Strategy, Protocol, Port, Target, Creation Time, and Action. Rules include allowing TCP port 7777 to 10.0.0.0/8 and TCP port 22 to 172.16.0.0/12.
- Bottom Screenshot (Baseline Scan):** Shows the "Baseline Scan" tab selected. It displays the results of a scan from March 28, 2022. The table shows findings across various security and system configuration items. For example, "Ensure SELinux is installed" is compliant (green), while "Ensure remote login warning banner is configured properly" is non-compliant (red).

图 5.3 主机资产扫描模块主要界面

(1) 主机资产界面：删除按钮对应 `Popover` 和 `Button` 元素，搜索框对应 `Input.Search` 元素，主机资产列表对应 `Table` 元素。当用户选择并点击删除按钮或点击操作中的删除按钮后，会将所有选择的主机资产编号作为参数传递给 `deleteAgent` 函数，然后向主机资产删除接口发送 `DELETE` 请求，响应成功之后则弹出删除成功消息框。当用户在搜索框中输入内容后敲击回车会设置请求 `query` 中的 `search` 字段，然后将 `query` 中的内容做为参数传递给 `getAgentList` 函数，然后向主机资产列表获取接口发送 `GET` 请求，响应成功之后向在列表中显示相应的内容。

(2) 性能监控界面：主磁盘监控和资源监控表单对应 `Row` 和 `Col` 元素，CPU 使用率图和内存使用率图是自定义基于 `echarts` 的 `DynamicLineGraph` 元素，通过设置请求参数 `pollingInterval` 可以使得每隔指定的时间向主机系统信息获取接口发送 `GET` 请求，然后使用请求获取到的数据对图表进行修改填充，使得页面可以动态显示主机资产的系统信息。

(3) 开放端口扫描界面：当用户点击开放端口扫描标签后会进入当前界面。端口类型和端口状态下拉框均对应 `Select` 元素，搜索框对应 `Input.Search` 元素，对外开放端口列表对应 `Table` 元素，点击详情弹出的侧边框是自定义基于 `Drawer` 的 `PortDetail` 元素。当用户选择下拉框或在搜索框填入内容后，会分别设置请求 `query` 中的 `port_type`、`port_status`、`search` 字段，然后将 `query` 中的内容作为参数传递给 `getPort` 函数，然后向主机对外开放端口接口发送 `GET` 请求，响应成功之后会在列表中显示相应的内容。

(4) 访问控制界面：当用户点击访问控制标签后进入当前界面。作用方向下拉框对应 `Select` 元素，新建规则按钮对应 `Button` 元素，安全组规则列表对应 `Table` 元素，操作中的删除按钮对应 `Button` 元素，用户点击新建规则按钮弹出的输入框是自定义基于 `Modal` 和 `Form` 元素的 `AddModal` 元素。当用户选择下拉框内容后，会设置请求 `query` 中的 `direction` 字段，然后将 `query` 中的内容作为参数传递给 `getSecgrp` 函数，然后向安全组规则列表获取接口发送 `GET` 请求，响应成功之后会在列表中显示相应的内容。当用户点击操作中的删除按钮后会将选择项的安全组规则编号传递给 `deleteSecgrp` 函数，然后向安全组规则删除接口发送 `DELETE` 请求，响应成功之后再次调用 `updateList` 函数更新安全组规则列表。当用户点击新增按钮并填入相应的内容后点击确定会将表单的内容做为参数传递给 `addSecgrp` 函数然后向安全组规则新增接口发送 `POST` 请求，响应成功之后再次调用 `updateList` 函数更新安全组规则列表。

(5) 基线扫描界面：当用户点击基线扫描标签后进入当前界面。扫描结果下拉框对应 `Select` 元素，扫描按钮对应 `Button` 元素，扫描结果详情对应 `Row` 和 `Col` 元素，扫描结果列表对应 `Table` 元素，操作中的建议按钮对应自定义基于 `Modal` 的 `BaselineDetail`

元素。当用户选择下拉框内容后会设置请求 query 中的 status 字段，然后将 query 中的内容作为参数传递给 `getBaseline` 函数，然后向基线扫描结果列表获取接口发送 GET 请求，响应成功后会在列表中显示相应的内容。当用户点击扫描按钮后会向开启基线扫描接口发送 POST 请求，响应成功后会更新扫描结果详情和列表内容。当用户点击操作中的忽略按钮或恢复按钮后会向将基线规则编号和主机资产编号传递给 `changeBaselineInfo` 函数，然后向基线扫描属性更新接口发送 PUT 请求，响应成功后会更新扫描结果详情内容。当用户点击建议按钮后会向基线扫描详情获取接口发送 GET 请求，然后将响应数据显示再侧边框中。

### 5.3.2 接口实现

登录注销模块的路由配置定义在 `nas-web/router/api/v1` 目录下的 `agent.go` 文件中，请求入口函数定义在 `nas-web/controller` 目录下的 `agent.go` 文件中，请求处理函数定义在 `nas-web/service` 目录下的 `agent.go` 文件中。所有接口都需要进行 JWT 校验。

(1) 主机列表获取接口：主机列表获取请求方式为 GET，请求路径为 `/v1/agent/`，请求入口函数为 `ListAgent`，该接口需要进行 FORM 形式的参数校验，请求处理函数为 `ListAgentHandler`。实现流程为：首先定义数据库查询语句 `bson.M{}` 并初始化 `{"is_delete": false}`，如果请求表单中的 `search` 不为空则分别针对 `agent_ip` 和 `hostname` 字段设置模糊查询语句，之后通过调用 MongoDB 提供的 `FindByLimitAndSkip` 函数结合请求表单中的 `page`, `page_size` 对 `tb_agent` 集合进行分页查询，将查询的结果存储到 `nas-common` 提供的公用的 `models.AgentHandlerInfo` 结构体中，然后将结构体中的数据存储到响应结构体 `ListAgentResp` 中，然后调用 `SendApiResponse` 将数据返回给网页面端。

(2) 主机系统信息获取接口：主机系统信息获取请求方式为 GET，请求路径为 `/v1/agent/info/system/`，请求入口函数为 `AgentSystemInfo`，该接口需要进行 FORM 形式的参数校验，请求处理函数为 `AgentSystemInfoHandler`。实现流程为：首先定义数据库查询语句 `bson.M{}` 并初始化 `{"hash_id": 请求表单中的主机资产编号}`，然后调用 MongoDB 提供的 `FindOne` 函数查询 `tb_agent_info` 集合并将结果保存到 `models.AgentHandlerInfoDetails` 结构体中，然后将结构体中的数据存储到响应结构体 `AgentSystemInfoResp` 中，然后调用 `SendApiResponse` 将数据返回给网页面端。

(3) 主机对外开放端口接口：主机对外开放端口请求方式为 GET，请求路径为 `/v1/agent/info/port/`，请求入口函数为 `AgentPortInfo`，该接口需要进行 FORM 形式的参数校验，请求处理函数为 `AgentPortInfoHandler`。实现流程为：首先定义数据库查询语句 `bson.M{}` 并初始化，若请求表单中的 `search` 不为空则分别针对 `port_infos` 字段中

的 `port`、`port_service` 进行元素匹配，然后调用 MongoDB 提供的 `FindOne` 函数查询 `tb_agent_info` 集合并将结果保存到 `models.AgentHandlerInfoDetails` 结构体中，然后将结构体中的数据存储到响应结构体 `AgentPortInfoResp` 中，最后调用 `SendApiResponse` 将数据返回给网页端。

(4) 代理程序下载接口：代理程序下载请求方式为 `POST`，请求路径为 `/v1/agent/download/`，请求入口函数为 `DownloadAgent`，该接口不需要进行参数校验，请求处理函数为 `DownloadAgentHandler`。实现流程为：将代理程序的绝对路径存储到变量 `path` 中，然后调用 `os.Open` 函数将代理程序打开，然后调用 `ioutil.ReadAll` 函数将文件内容转为比特流，然后设置响应请求的头部字段 `Content-Disposition` 为 `attachment; filename=agent-1.0.0.zip` 和 `Content-Type` 为 `application/octet-stream`，然后将调用 `Write` 函数将比特流写入响应请求中，然后调用 `SendApiResponse` 将成功消息返回给网页端。

(5) 主机资产删除接口：主机资产删除请求方式为 `DELETE`，请求路径为 `/v1/agent/`，请求入口函数为 `DeleteAgent`，该接口需要进行 JSON 形式的参数校验，请求处理函数为 `DeleteAgentHandler`。实现流程为：首先设置查询语句为 `bson.M{}` 并初始化 `{"hash_id":bson.M{"$in":请求体中的主机资产编号切片}}`，然后调用 MongoDB 提供的 `RemoveAll` 函数操作 `tb_agent` 集合删除切片中的所有主机资产，采用这种方式即可实现批量删除的功能，然后调用 `SendApiResponse` 将成功消息返回给网页端。

(6) 安全组规则新增接口：安全组规则新增请求方式为 `POST`，请求路径为 `/v1/agent/secgrp/`，请求入口函数为 `AddAgentSecGrpRule`，该接口需要进行 JSON 形式的参数校验，请求处理函数为 `AddAgentSecGrpRuleHandler`。实现流程为：首先对请求体中的数据进行校验，如果 `port` 超出指定的范围或 CIDR 不符合相应的正则匹配则会返回数据无效的响应，然后通过请求体中的 `rule_id` 和 `agent_id` 来查询数据库判断是否正在新增重复的规则，若重复则返回重复新增的响应，然后设置发送内容 `ForwardAgentActionReq` 并将其中的行为标记设置为 `Action_ADDRULE`，然后调用 RPC 接口函数 `ForwardAgentAction` 将需要新增的安全组规则发送给业务逻辑端的 `forward` 服务，如果发送失败会返回发送失败的响应，然后会将新增的安全则规则插入到 `tb_srcgrp` 集合中，然后调用 `SendApiResponse` 返回成功的响应给网页端。

(7) 安全组规则删除接口：安全组规则删除请求方式为 `DELETE`，请求路径为 `/v1/agent/secgrp/`，请求入口函数为 `DeleteAgentSecGrpRule`，该接口需要进行 JSON 形式的参数校验，请求处理函数为 `DeleteAgentSecGrpRuleHandler`。实现流程为：首先通过请求体中的数据查询调用 MongoDB 提供的 `FindOne` 函数查询 `tb_secgrp` 集合，若不存在响应的规则，则返回数据不存在的响应，然后设置发送内容

ForwardAgentActionReq 并将其中的行为标记设置为 Action\_DELETERULE，然后调用 RPC 接口函数 ForwardAgentAction 将需要删除的安全组规则发送给业务逻辑端的 forward 服务，若发送失败则返回发送失败的响应，然后再删除 tb\_secgrp 集合中相应的安全组规则，然后调用 SendApiResponse 返回成功的响应给网页端。

(8) 安全组规则列表获取接口：安全组规则列表获取请求方式为 GET，请求路径为 /v1/agent/secgrp/，请求入口函数为 ListAgentSecGrp，该接口需要进行 FORM 形式的参数校验，请求处理函数为 ListAgentSecGrpHandler。实现流程为：首先设置数据库查询语句 bson.M{} 并初始化为 {"agent\_id": 请求表单中的主机资产编号}，然后设置 {"direction": 请求表单中的作用方向}，然后调用 MongoDB 提供的 FindSortByLimitAndSkip 函数结合请求表单中的 page、page\_size 对 tb\_srcggrp 集合进行分页查询并将结果保存到 []models.SecGrp 切片中，然后将查询结果存储到响应结构体 ListAgentSecGrpResp 并调用 SendApiResponse 将数据返回给网页端。

(9) 开启基线扫描接口：开启基线扫描请求方式为 POST，请求路径为 /v1/agent/baseline，请求入口函数为 StartBaselineScan，该接口需要进行 JSON 形式的参数校验，请求处理函数为 StartBaselineScanHandler。实现流程为：首先设置发送内容 ForwardAgentActionReq 并将其中的行为标记设置为 Action\_STARTCISCHECK，然后调用 RPC 接口函数 SendSecGrpRule 将启动基线扫描的主机资产编号发送给 forward 服务，若发送失败则返回发送失败的响应，然后调用 OpenWait 函数订阅 Redis 缓存标记检测是否扫描结束，然后调用 SendApiResponse 返回成功的响应给网页端。

(10) 基线扫描结果获取接口：基线扫描结果请求方式为 GET，请求路径为 /v1/agent/baseline/，请求入口函数为 ListAgentBaseline，该接口需要进行 FORM 形式的参数校验，请求处理函数为 ListAgentBaselineHandler。实现流程为：首先调用 MongoDB 提供的 FindCount 函数查询 tb\_scan\_cis 集合中相应主机资产编号的扫描结果数 id，id 即代表最近一次扫描结果的序号，利用此序号再进行精确查找，然后再设置精确查找的查询语句为 bson.M{"agent\_id": 主机资产编号, "id": id}，然后调用 MongoDB 提供的 FindByLimitAndSkip 函数结合请求表单中的 page、page\_size 进行分页查询并将结果保存到 models.CisScanResultItem 结构体中，然后再将结果整合到响应结构体 ListAgentBaselineResp 中，然后调用 SendApiResponse 将数据返回给网页端。

(11) 基线扫描属性更新接口：基线扫描属性更新请求方式为 PUT，请求路径为 /v1/agent/baseline/，请求入口函数为 UpdateAgentBaseline，该接口需要进行 JSON 形式的参数校验，请求处理函数为 UpdateAgentBaselineHandler。实现流程为：首先查询数据库 tb\_scan\_cis 集合的数量 count，然后利用该 count 和请求体中的基线规则编号和主机资产编号构造查询语句，然后调用 MongoDB 提供的 FindOne 函数对

`tb_scan_cis_results` 进行查询，利用查询结果构造更新语句，然后调用 MongoDB 提供的 `Update` 函数分别对集合 `tb_scan_cis` 和 `tb_scan_cis_results` 进行更新，然后调用 `SendApiResponse` 将成功的响应返回给网网页端。

(12) 基线扫描详情获取接口：基线扫描详情获取请求方式为 GET，请求路径为 `/v1/agent/baseline/info/`，请求入口函数为 `GetBaselineInfo`，该接口需要进行 FORM 形式的参数校验，请求处理函数为 `GetBaselineInfoHandler`。实现流程为：首先使用请求表单中的规则编号构造查询语句，然后调用 MongoDB 提供的 `FindOne` 函数对 `tb_repo_cis` 集合进行查询并将结果保存到 `models.TbRepoCis` 结构体中，然后再将结构体中的数据整合到响应结构体 `GetBaselineInfoResp`，然后调用 `SendApiResponse` 将响应数据返回给网网页端。

### 5.3.3 代理程序实现

代理程序设计框架基于业务逻辑端的 App 框架进行实现，会设计到与 collect 服务的交互，在程序初始化过程中需要调用 `CheckRoot` 函数检测运行权限，然后调用 `GetRpcClientCollect().Init()` 函数对 collect 服务的 RPC 客户端进行初始化，然后调用 `InitSecGrp` 函数对安全组规则链进行初始化。该程序需要连接 Redis 数据库以实现基线扫描同步功能，因此需要调用 `NewRedisPlugin` 函数将 Redis 作为插件加载进程序中。该程序中有两部分功能，一是将收集到的主机信息以 RPC 的方式传输给 collect 服务，二是订阅 Kafka 指定的 topic 以响应相应的指令。为了使这两个功能便于管理，使用 `Server` 结构体对两者进行封装，同时提供 `NewServer` 函数用于初始化操作，`Stop` 函数用于停止服务操作，另外，服务初始化操作只需要执行一次，因此调用 `NewSingleton` 函数以单例模式调用 `NewServer` 函数进行服务的初始化操作。

信息收集具有定时性，因此定义了一个 `CollectInfoService` 结构体，其中包含一个 `*time.Ticker` 定时器，同时提供 `NewCollectInfoService` 函数用于初始化，该函数会在 `NewServer` 函数中被调用，另外提供 `Stop` 函数用于停止，该函数会在 `Server Stop` 函数中被调用。真正的运行函数为 `Run` 函数，其逻辑为定时死循环，其中包括两个部分，一是收集信息，二是将信息发送给 collect 服务。通过调用 `CollectData` 函数将收集到的信息存储到 `CollectAgentBasicInfoReq` 结构体中，然后调用 `SendData` 函数将结构体中的内容通过 RPC 接口函数 `SendAgentBasicInfoToServer` 发送给 collect 服务。

Kafka 服务会订阅 `forward` 推送的 topic，该 topic 以主机资产的 MAC 地址 MD5 作为区分，另外，使用 Kafka 的 group 功能用来限制同组中只能有一个 Kafka 服务能够获取消息队列中的数据。首先定义一个结构体 `MqKafkaService`，其中包括一个 `*queue.QueueKafka` 结构体，同时提供 `NewMqKafkaService` 函数用于初始化，该函数

会在 `NewServer` 中被调用，该函数中会真正调用 `queue.NewQueueKafka` 函数进行初始化，同时设定数据接受后执行函数 `NewAgentOperationReceiver`。在该执行函数中，通过调用 `json.Unmarshal` 函数将接受数据反序列化到 `models.SecGrpRule` 结构体中，然后根据结构体中的 `Action` 字段来进行行为的判断，如果值是 0 或 1，则执行安全组规则相关的操作，其中会涉及到第三方库 `github.com/coreos/go-iptables/iptables` 的调用，如果值是 2，会进行基线扫描，其中包括两个部分，一是执行编写的扫描脚本 `scan.sh` 然后将扫描结果存储到 `collect.CollectAgentCisScanResultReq` 结构体中，二是将结构体中的内容通过 `SendAgentCisScanResultToServer` 函数发送给 `collect` 服务。

### 5.3.3 collect 服务实现

`collect` 服务也是基于业务逻辑端的 App 框架进行实现的，其主要包括两部分功能，一是处理主机资产上报的数据，二是处理主机上报的基线扫描结果。这两部分功能均对应服务端 RPC 接口函数的重写。首先是 RPC proto 文件的编写，在 `collect.proto` 文件定义 `service Collect`，然后在函数体中分别定义 `rpc CollectAgentBasicInfo(stream){}` 用于流式收集并处理主机上报的信息和 `rpcCollectAgentCisScanResult(){}` 用于收集并处理主机上报的基线扫描结果。在 `collect` 服务中分别对这两个函数接口进行了重写，其主要内容是将数据分别进行整理然后写入 MongoDB 数据库中。

### 5.3.3 forward 服务实现

`forward` 服务也是基于业务逻辑端的 App 框架进行实现的，其主要功能是将收集到的数据进行整理然后发布到 kafka 指定分组相应的以主机资产为区别的 topic 中，这一部分功能在重写的 RPC Server 函数中进行实现。首先是 proto 文件的编写，在 `forward.proto` 文件中定义 `service Forward`，然后在函数体中定义 `rpc ForwardAgentAction(){}` 用于转发代理程序执行行为。接下来是服务启动，为了避免服务初始化多次，调用 `NewSingleton` 函数以单例模式执行 `NewServer` 函数进行服务初始化，由于 `forward` 服务会设计到 Kafka 发布功能，因此也需要以单例模式调用 `NewKafkaDispatcher` 函数进行初始化。重写的 RPC 函数接口主要内容是将接受到的数据存储到 `models.SecGrpRule` 结构体中，然后调用 `NewKafkaContext` 函数设置 topic，然后调用 `Dispatch` 函数将反序列化的结构体数据发布到 Kafka 消息队列中。

## 5.4 网络资产扫描模块实现

网络资产扫描模块界面主要包括网络资产扫描界面、扫描结果汇总界面和扫描 IP 详情界面，如图 5.4 所示。接口主要包括新建网络资产扫描接口、获取网络资产扫描列

## 基于 ZMap 的网络资产扫描系统设计与实现

表接口、删除网络资产扫描项接口、获取预扫结果接口、获取深扫结果接口，该模块涉及到与业务逻辑端 monitor 服务、zmap 服务、nmap 服务的交互。

### 5.4.1 主要界面

The screenshot displays two pages of the network asset scan system:

- Top Page (Scan Job List):** Shows a table of scan jobs. Each job entry includes the IP range, start time, scan type, status, end time, and operations (Delete | Details). There are 6 entries listed, all marked as 'Scan Success'.

扫描IP域	开始时间	扫描类型	扫描状态	结束时间	操作
47.117.1.7/24	2022/5/7 17:59:56	普通扫描	扫描成功	2022/5/8 14:23:14	<a href="#">删除</a>   <a href="#">详情</a>
47.117.1.7/24	2022/5/7 15:08:15	普通扫描	扫描成功	2022/5/7 15:13:19	<a href="#">删除</a>   <a href="#">详情</a>
47.117.1.7/24	2022/5/6 20:35:58	普通扫描	扫描成功	2022/5/6 20:37:46	<a href="#">删除</a>   <a href="#">详情</a>
47.117.1.7/24	2022/5/6 16:41:23	普通扫描	扫描成功	2022/5/6 16:50:07	<a href="#">删除</a>   <a href="#">详情</a>
47.117.1.7/24	2022/5/6 16:41:17	普通扫描	扫描成功	2022/5/6 16:48:31	<a href="#">删除</a>   <a href="#">详情</a>
47.117.1.7/24	2022/5/6 16:41:09	普通扫描	扫描成功	2022/5/6 16:47:27	<a href="#">删除</a>   <a href="#">详情</a>
- Bottom Page (Scan Result Summary):** Shows a summary of the scan results. It includes a gauge chart for success rate (100%), a table for scan details, and two bar charts for port distribution and service distribution.

扫描成功率	扫描IP域	开始时间	结束时间
100%	47.117.1.7/24	2022/5/7 17:59:57	2022/5/8 14:23:14

深度扫描用时	扫描类型	扫描选项
00:09:27	普通扫描	<input checked="" type="checkbox"/> OS <input type="checkbox"/> Service <input type="checkbox"/> Script <input type="checkbox"/> Trace

Port Distribution Chart (Port Number vs. Count):

端口号	数量
21	55
111	32
1935	12
3389	10
5252	5
8022	5
8088	5
8888	10
9011	5
10003	2
27017	2

Service Distribution Chart (Service Name vs. Count):

服务名	数量
afs3-callback	2
documentum	2
http-proxy	58
mysql	22
ssh	55

## 基于 ZMap 的网络资产扫描系统设计与实现

The figure consists of three vertically stacked screenshots of a network asset management system's asset scanning module. The top screenshot shows a table of scanned ports with the following data:

端口号	协议类型	服务名称	TTL	服务状态	操作
22	tcp	ssh	41	open	服务详情
80	tcp	http	43	open	服务详情
443	tcp	http	41	open	服务详情
3306	tcp	mysql	41	open	服务详情
5252	tcp	movaz-ssc	44	open	服务详情
2222	tcp	ssh	45	open	服务详情
27017	tcp	mongod	42	open	服务详情

The middle screenshot shows a route tracing diagram for port 3306, starting from 10.108.21.200 and ending at 172.31.1.10, with various intermediate IP addresses shown along the path.

The bottom screenshot shows a large blue circle with the text "Linux 4.4" inside it.

图 5.4 网络资产扫描模块主要界面

(1) 网络资产扫描界面：当用户点击左侧导航栏资产管理下的网络资产时进入当前界面。新建扫描按钮和删除按钮对应 `Popover` 和 `Button` 元素，搜索框对应 `Input.Search` 元素，扫描列表对应 `Table` 元素。当用户进入此界面后，会自动向获取网络资产扫描列表接口发送 `GET` 请求，响应成功后会更新扫描列表的内容。当用户点击新建扫描按钮会弹出自定义的基于 `Modal` 和 `Form` 元素的 `AddScanModal` 元素的表单，填写相应的内容后点击确定会调用 `validateFields` 函数校验表单数据是否合规，然后将

表单数据通过参数传递给 `addScan` 函数，然后向网络资产扫描接口发送 `POST` 请求，响应成功后会调用 `updateList` 函数更新扫描列表。当用户选择删除或点击操作中的删除按钮后，会将选择的扫描项编号 `scan_ids` 作为参数传递给 `deleteScan` 函数，然后向删除网络资产扫描项接口发送 `DELETE` 请求，响应成功后会调用 `updateList` 函数更新扫描列表的内容。当用户点击操作中的详情按钮后进入扫描结果界面。

(2) 扫描结果汇总界面：该界面扫描结果汇总对应 `Row` 和 `Col` 元素，端口分布图和服务分布图是自定义基于 `echarts` 的 `BarGraph` 元素，已扫描 IP 列表对应 `Collapse`、`Panel` 和 `List` 元素。当用户进入此界面后，会自动向获取预扫结果接口发送 `GET` 请求，响应成功后将数据分别置入相应的元素内。已扫描 IP 列表中会列举出所有扫描成功的 IP，该 IP 可点击，点击后会进入扫描 IP 详情界面。

(3) 扫描 IP 详情界面：该界面上包括三个 `Tabs` 元素分别对应三个子界面，分别是端口服务界面、路由追踪界面和系统探测界面，其中，端口服务界面是默认界面，当用户进入此界面后，会自动向获取深扫结果接口发送 `GET` 请求，响应成功后会将数据分别置入相应的元素内。端口服务界面中的提示信息对应 `Divider` 元素，端口服务列表对应 `Table` 元素，当用户点击服务详情按钮会弹出侧边框，其中主要包括端口服务的详细信息和脚本探测信息。路由追踪界面中的路径图是自定义基于 `echarts` 的 `PathGraph` 元素。系统探测界面中的饼状图是自定义基于 `echarts` 的 `PieGraph` 元素。

#### 5.4.2 接口实现

网络资产扫描模块的路由配置定义在 `nas-web/router/api/v1` 目录下的 `scan.go` 文件中，请求入口函数定义在 `nas-web/controller` 目录下的 `scan.go` 文件中，请求处理函数定义在 `nas-web/service` 目录下的 `scan.go` 文件中。所有接口都需要进行 JWT 校验。

(1) 新建网络资产扫描接口：请求方式为 `POST`，请求路径为 `/v1/scan/`，请求入口函数为 `AddScan`，该接口需要进行 `JSON` 形式的参数校验，请求处理函数为 `AddScanHandler`。实现流程为：首先将请求体中的数据存储到 `models.ScanStatusDesc` 结构体中，然后调用 `MongoDB` 提供的 `Insert` 方向将结构体中的数据插入到 `tb_scan_status_desc` 集合中，然后将扫描数据存储到 `models.ScanInfo` 结构体中，然后调用 `json.Marshal` 将数据转化为比特流，然后调用 `Kafka SetContext` 函数和 `Dispatch` 函数将比特流发布到 `common.KafkaTopicZmapScanData topic` 中，然后返回成功的响应给网页端。

(2) 获取网络资产扫描列表接口：请求方式为 `GET`，请求路径为 `/v1/scan/`，请求入口函数为 `ListScan`，该接口需要进行 `FORM` 形式的参数校验，请求处理函数为 `ListScanHandler`。实现流程为：首先使用请求 `query` 中的字段构造数据库查询语句，

然后调用 MongoDB 提供的 `FindSortByLimitAndSkip` 函数结合请求 `query` 中的 `page`、`page_size` 字段对 `tb_scan_status_desc` 集合进行分页查询并将结果保存到 `[]models.ScanStatusDesc` 切片中，然后将数据整合到响应结构体 `ListScanResp` 中，然后调用 `SendApiResponse` 函数将数据返回给网页端。

(3) 删除网络资产扫描项接口：请求方式为 `DELETE`，请求路径为 `/v1/scan/`，请求入口函数为 `DeleteScan`，该接口需要进行 `JSON` 形式的参数校验，请求处理函数为 `DeleteScanHandler`。实现流程为：循环调用 MongoDB 提供的 `FindOne` 函数查询 `tb_scan_status_desc` 集合是否存在相应的扫描项编号，若存在，则调用 `RemoveAll` 函数删除，循环结束后则调用 `SendApiResponse` 返回成功的响应给网页端。

(4) 获取预扫结果接口：请求方式为 `GET`，请求路径为 `/v1/scan/getfirst/`，请求入口函数为 `GetFirstScanResult`，该接口需要进行 `FORM` 形式的参数校验，请求执行函数为 `GetFirstScanResultHandler`。实现流程为：首先使用请求 `query` 中的扫描项编号构造查询语句，然后调用 MongoDB 提供的 `FindOne` 函数对 `tb_first_scan_result` 集合和 `tb_deep_scan_result` 集合进行查询并将结果保存到 `models.FirstScanIpResult` 结构体和 `models.DeepScanIpResult` 结构体，然后将查询结果整合到响应结构体 `GetFirstScanResultResp` 中，然后调用 `SendApiResponse` 将数据返回给网页端。

(5) 获取深扫结果结果：请求方式为 `GET`，请求路径为 `/v1/scan/ip/`，请求入口函数为 `GetScanIpResult`，该接口需要进行 `FORM` 进行的参数校验，请求执行函数为 `GetScanIpResultHandler`。实现流程为：首先利用请求 `query` 中的扫描项编号构造查询语句，然后调用 MongDB 提供的 `FindOne` 函数查询 `tb_deep_scan_result` 集合并将结果保存到 `models.DeepScanIpResult` 结构体中，然后利用请求 `query` 中的 IP 来对数据进行过滤并将结果整合到响应结构体 `GetScanIpResultResp` 中，然后调用 `SendApiResponse` 将数据返回给网页端。

#### 5. 4. 3 monitor 服务实现

`monitor` 服务存储于结构体 `Server`，其中包含 `*MqKafkaService` 对象，另外提供了 `NewServer` 函数和 `Stop` 函数用于初始化服务和停止服务，为了使服务只初始化一次，需要调用 `NewSingleton` 函数以单例模式执行 `NewServer` 函数，该服务会使用到 `Zmap` RPC 接口客户端函数，因此需要在 `App` 初始化函数中调用 `GetRpcClientZmap().Init()` 函数进行初始化，`*MqKafkaService` 在初始化时会调用 `NewQueueKafka` 函数对 `common.KafkaTopicZmapScanData topic` 进行订阅，回调函数为 `NewZmapScanDataReceiver`，该函数的主要功能是对扫描项数据进行整合，然后调用 `Zmap` RPC 接口函数 `SendScanIpToZmap` 将扫描项数据发送给 `zmap` 服务。

#### 5.4.4 zmap 服务实现

zmap 服务会使用到 Kafka 的发送功能，因此需要调用 NewSingleton 函数以单例模式执行 NewKafkaDispatcher 函数来初始化 Kafka 客户端。zmap 服务具体功能对应 Zmap RPC Server 接口函数的重写，首先是 RPC proto 文件的编写，在 zmap.proto 文件中定义 service ZmapSrv，然后在函数体中定义 rpc DeliverScanIp(){} 用于扫描项数据传递，在 zmap 服务中对这个函数接口的服务端进行了重写。其主要内容是调用 exec.Command 函数结合扫描项数据中的内容执行 zmap 命令并将扫描结果存储到相应的文件中，然后调用 os.Open 函数读取扫描结果并整合到 models.FirstScanIpResult 结构体中，然后使用 MongoDB 提供的 Insert 函数将数据插入到 tb\_first\_scan\_result 集合中，然后调用 Update 函数修改 tb\_scan\_status\_desc 集合中的扫描状态，然后会调用 Kafka 中定义的 Dispatch 函数将扫描信息发布到 common.KafkaTopicZmapScanResultData topic 中，然后返回。

#### 5.4.3 nmap 服务实现

nmap 服务存储于结构体 Server，其中包含 \*MqKafkaService 对象，另外提供了 NewServer 函数和 Stop 函数用于初始化服务和停止服务，为了使服务只初始化一次，需要调用 NewSingleton 函数以单例模式执行 NewServer 函数，\*MqKafkaService 在初始化时会调用 NewQueueKafka 函数对 common.KafkaTopicZmapScanResultData topic 进行订阅，回调函数为 NewNmapScanReceiver，该函数的主要功能是首先对获取到的数据进行反序列化，根据扫描项的数据结合第三方库 [github.com/Ullaakut/nmap/v2](https://github.com/Ullaakut/nmap/v2) 进行 nmap 深度扫描并将扫描结果存储到 models.DeepScanIpResult 结构体中，然后调用 MongoDB 提供的 Insert 函数将扫描结果插入到 tb\_deep\_scan\_result 集合中并调用 Update 函数更新 tb\_scan\_status\_desc 集合中的扫描状态。

### 5.5 用户管理模块实现

用户管理模块界面包括账户信息界面和用户管理界面，如图 5.5 所示。接口包括添加用户接口、删除用户接口、编辑账户接口、获取用户列表接口、获取账户信息接口、修改账户密码接口、修改用户登录状态接口、重置用户密码接口，该模块不涉及与业务逻辑端的交互。

#### 5.5.1 主要界面

## 基于 ZMap 的网络资产扫描系统设计与实现

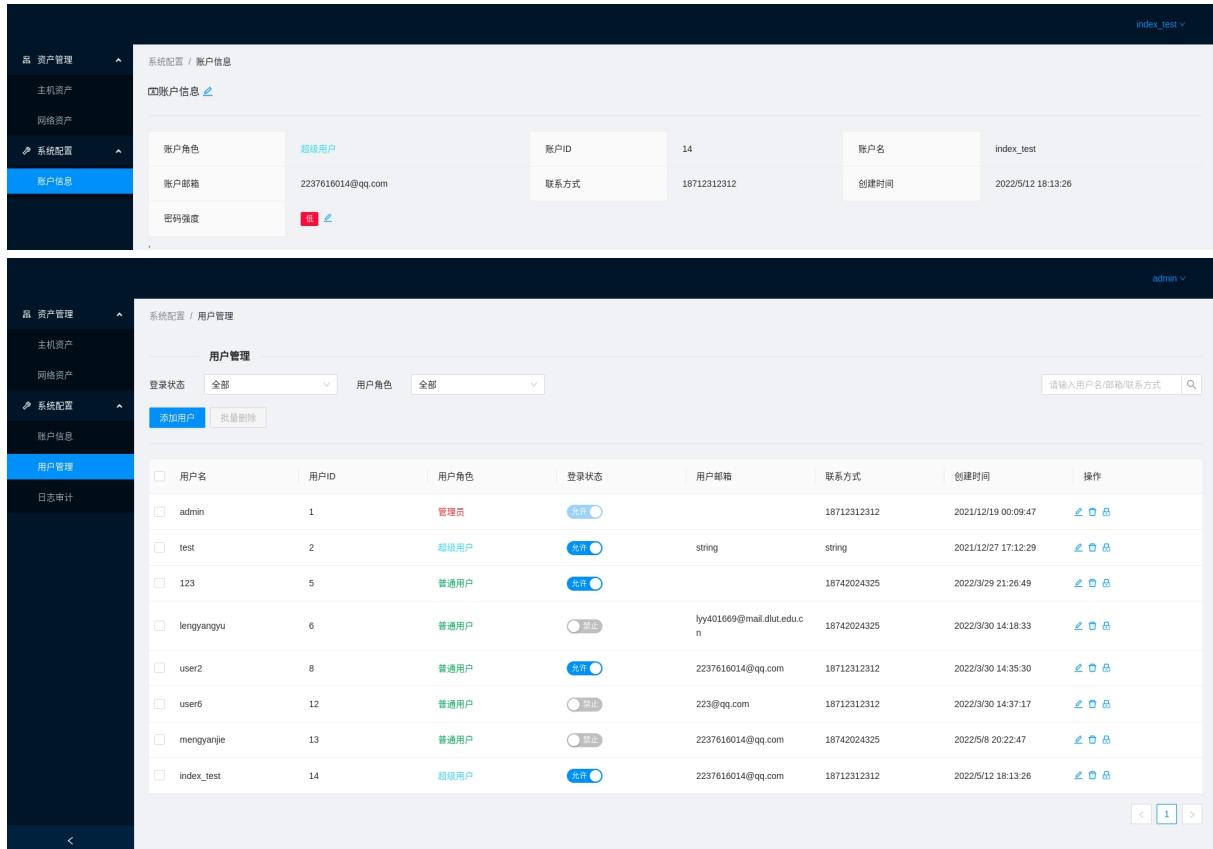


图 5.5 用户管理模块主要界面

(1) 账户信息界面：当用户点击左侧导航栏中账户信息后会进入当前界面。该界面中的账户信息对应 `Descriptions` 和 `Descriptions.Item` 元素，当用户点击账户信息或密码强度旁边的编辑图标会弹出自定义基于 `Modal` 和 `Form` 的 `EditModal` 元素或 `ChangePasswdModal` 元素。当进入此界面后会自动向获取账户信息接口发送 `GET` 请求，响应成功后将数据分别置入相应的元素。当用户填写完 `EditModal` 中的表单内容后点击确定会调用 `validateFields` 函数校验表单信息然后传递给 `editAccountInfo` 函数向编辑账户接口发送 `PUT` 请求，响应成功后调用 `updateList` 函数更新账户信息。当用户填写完 `ChangeOasswdModal` 中的表单内容后会调用 `validateFields` 函数校验扁担信息然后传递给 `changeAccountPassword` 函数向修改账户密码接口发送 `PUT` 请求，响应成功成功后也会调用 `updateList` 函数更新账户信息。

(2) 用户管理界面：当用户为管理员时点击左侧导航栏中的用户管理会进入当前界面。该界面中的登录状态和用户角色下拉框对应 `Select` 元素，搜索框对应 `Input.Search` 元素，添加用户按钮和批量删除按钮对应 `Button` 元素，用户列表对应 `Table` 元素，登录状态开关对应 `OnlineStatus` 元素。当进入此界面后会自动向获取用户列表接口发送 `GET` 请求，响应成功后将数据填充进 `Table` 中。当管理员点击添加用户按钮后回

弹出自定义基于 Modal 和 Form 元素的 AddUserModal 元素，填写完表单内容后点击确定会调用 validateFields 函数检验表单信息然后传递给 addUser 函数向添加用户接口发送 POST 请求，响应成功后会调用 updateList 函数更新用户列表，选择点击批量删除按钮或操作中的删除图标会向删除用户接口发送 DELETE 请求，点击操作中的编辑图标或锁图标会向编辑账户信息接口和重置用户密码接口发送 PUT 请求和 POST 请求，点击用户列表中的登录状态开关会向修改用户登录状态接口发送 PUT 请求。

### 5.5.2 接口实现

用户管理模块的路由配置定义在 `nas-web/router/api/v1` 目录下的 `user.go` 文件中，请求入口函数定义在 `nas-web/controller` 目录下的 `user.go` 文件中，请求处理函数定义在 `nas-web/service` 目录下的 `user.go` 文件中。所有接口都需要进行 JWT 校验。

(1) 添加用户接口：请求方式为 POST，请求路径为 `/v1/user/`，请求入口函数为 `AddUser`，该接口需要进行 JSON 形式的参数校验，请求处理函数为 `AddUserHandler`。实现流程为：首先使用 `Context` 结构体中存储的用户类型判断当前用户是否有权限增添用户，然后调用 `webutils` 中的 `Compare` 函数判断密码是否相同，然后调用 `MongoDB` 中的 `FindAll` 函数查询 `tb_user` 集合判断用户是否存在，然后调用 `webutils` 中的 `GetPasswordStrength` 函数判断密码强度，然后将新增用户信息保存到 `models.User` 结构体中，然后调用 `MongoDB` 提供的 `Insert` 函数将用户数据插入到 `tb_user` 集合，然后调用 `SendApiResponse` 返回成功的响应。

(2) 删除用户接口：请求方式为 DELETE，请求路径为 `/v1/user/`，请求入口函数为 `DeleteUser`，该接口需要进行 JSON 形式的参数校验，请求处理函数为 `DeleteUserHandler`。实现流程为：首先判断当前用户是否有权限删除用户，然后调用 `MongoDB` 提供的 `RemoveAll` 函数依次删除 `tb_user` 集合中的用户信息（管理员无法删除自己的账户），然后调用 `SendApiResponse` 返回成功的响应。

(3) 编辑账户接口：请求方式为 PUT，请求路径为 `/v1/user/`，请求入口函数为 `EditUser`，该接口需要进行 JSON 形式的参数校验，请求执行函数为 `EditUserHandler`。实现流程为：首先判断用户权限，管理员任意编辑，其他用户只能编辑自己的账户，然后使用请求体中的数据构造更新语句，然后调用 `MongoDB` 提供的 `Update` 函数对 `tb_user` 集合进行更新，然后调用 `SendApiResponse` 返回成功的响应。

(4) 获取用户列表接口：请求方式为 GET，请求路径为 `/v1/user/`，请求入口函数为 `ListUser`，该接口需要进行 FORM 形式的参数校验，请求执行函数为 `ListUserHandler`。实现流程为：首先使用请求结构体 `ListUserReq` 中的数据构造查询语句，然后调用 `MongoDB` 提供的 `FindByLimitAndSkip` 函数结合 `ListUserReq` 中的 `page`、`page_size` 字

段对 `tb_user` 集合进行分页查询并将查询结果存于 `[]models.User` 切片中，然后将结果整合到响应结构体 `ListUserResp` 中，然后调用 `SendApiResponse` 将数据返回网页端。

(5) 获取账户信息接口：请求方式为 `GET`，请求路径为 `/v1/user/info/`，请求入口函数为 `GetUserInfo`，该接口不需要进行参数校验，请求执行函数为 `GetUserInfoHandler`。实现流程为：首先使用 `Context` 结构体中的用户 ID 构造查询语句，然后调用 `MongoDB` 提供的 `FindOne` 函数查询 `tb_user` 集合并将结果保存到 `models.User` 结构体，然后将结果整合到响应结构体 `GetUserInfoResp` 中，然后调用 `SendApiResponse` 将数据返回给网页端。

(6) 修改账户密码接口：请求方式为 `PUT`，请求路径为 `/v1/user/passwd/`，请求入口函数为 `UpdateUserPasswd`，该接口需要进行 `JSON` 形式的参数校验，请求执行函数为 `UpdateUserPasswdHandler`。实现流程为：首先判断用户权限，所有的用户都只能修改自己的密码，然后判断请求结构体中的两次新密码是否相同，然后使用用户编号构造查询语句，然后调用 `MongoDB` 提供的 `FindOne` 函数查询 `tb_user` 集合并将结果保存到 `models.User` 结构体中，然后调用 `CheckPassword` 函数判断请求结构体中的密码和当前密码是否相同，然后调用 `MongoDB` 提供的 `Update` 函数结合新密码更新 `tb_user` 集合，然后调用 `SendApiResponse` 返回成功的响应。

(7) 修改用户登录状态接口：请求方式为 `PUT`，请求路径为 `/v1/user/status/`，请求入口函数为 `UpdateUserStatus`，该接口需要进行 `JSON` 形式的参数校验，请求执行函数为 `UpdateUserStatusHandler`。实现流程为：首先判断用户权限，只有管理员可以修改用户登录状态，然后使用请求结构体中的登录状态结合和用户编号构造查询语句和更新语句，然后调用 `MongoDB` 提供的 `Update` 函数更新 `tb_user` 集合，然后调用 `SendApiResponse` 返回成功的响应。

(8) 重置用户密码接口：请求方式为 `POST`，请求路径为 `/v1/user/reset_passwd/`，请求入口函数为 `ResetPasswd`，该接口需要进行 `JSON` 形式的参数校验，请求执行函数为 `ResetPasswdHandler`。实现流程为：首先判断用户权限，只有管理员可以重置用户密码，然后使用请求结构体中的重置密码和用户编号构造更新语句和查询语句，然后调用 `MongoDB` 提供的 `Update` 函数更新 `tb_user` 集合，然后调用 `SendApiResponse` 返回成功的响应。

## 5.6 阿里云服务器部署

阿里云服务器部署分为两大部分，在服务器 1 上，首先是 `Docker` 和 `Nginx` 的安装与配置，然后是运行 `MongoDB`、`Redis`、网页接口端容器，最后是安排网页端服务代码

存放和 **agent** 程序存放。在服务器 2 上，首先是 Docker 和 Supervisor 的安装与配置，然后是运行 Etcd、Kafka 容器，然后是运行业务逻辑端服务。

首先是 Docker 安装与配置。首先使用 `yum-config-manager` 命令添加阿里云 Docker 仓库，然后使用 `yum install` 命令安装 Docker，然后使用 `systemctl` 命令运行 Docker 服务器即可。本系统使用的 Docker 是 3.3.1 版本。

接下来 Nginx 的安装与配置。在服务器 1 上，首先运行 `yum install` 安装 Nginx，然后运行 `systemctl` 命令运行 Nginx。Nginx 的配置文件是 `/etc/nginx/nginx.conf`，通过在其中的 `http` 体中新增 `server` 体即可修改配置，使得网页请求和网页分别走不同的反向代理，然后运行 `systemctl reload` 命令重新加载配置即可。

接下来是 MongoDB、Redis、Etcd、Kafka 容器部署。MongoDB 使用的镜像是 Docker 镜像仓库的 `yowoo/my-mongo:latest`，Redis 使用的镜像是 Docker 镜像仓库的 `redis:latest`。Etcd 使用的镜像是 `quay.io/coreos/etcd:v3.4.15`，Kafka 使用的镜像是 `wurstmeister/kafka:latest`。通过配置 `docker-compose.yaml` 的 `image`、`ports`、`volumes`、`command`、`networks` 等字段后运行 `docker-compose up -d` 命令，即可分别在服务器 1 和 2 上运行这四个容器。

接下来是网页接口端容器部署。首先在本地环境进入 `nas-web/deployment` 目录，运行 `make` 命令，然后返回上级目录运行 `docker build` 命令构建网页接口端镜像，然后运行 `docker images` 检查是否构建成功，然后运行 `docker save` 命令将网页接口端镜像打成 `tar` 包，然后运行 `scp` 命令将 `tar` 包发送给服务器 1。进入服务器 1，运行 `docker load` 命令将 `tar` 包解压成镜像，然后运行 `docker run` 命令运行网页接口端容器。

接下来是网页端服务代码存放。首先在本地环境进入 `nas-ui` 目录，运行 `yarn build` 命令会自动生成 `dist` 目录，其中包含了网页端服务的静态文件，然后运行 `scp` 命令将 `dist` 目录发送给服务器 1 下的指定目录。进入服务器 1，修改 Nginx 请求定向的配置即可。

接下来是 **agent** 程序存放。首先在本地环境进入 `nas-daemon/deployment` 目录，然后运行 `make` 命令，然后使用 `zip` 命令将 **agent** 程序和相关文件压缩，然后运行 `scp` 命令将压缩文件发送给服务器 1 下的指定目录即可。

接下来是 Supervisor 的安装与配置。在服务器 2 上，运行 `yum install` 命令安装 `supervisor`，然后运行 `systemctl` 命令运行 `supervisord`，通过在 `/etc/supervisord.d` 目录下新增 `.ini` 配置文件的形式来运行 `supervisord` 进程管理。

接下来是运行业务逻辑端服务。首先进入本地环境 `nas-daemon/deployment` 目录，然后运行 `make` 命令即可编译生成所有服务程序，然后运行 `zip` 命令将服务程序和相关文件压缩，然后运行 `scp` 命令将压缩包发送给服务器 2，然后再在 `/etc/supervisord.d/` 目录下分别编写各个服务的 `.ini` 文件，然后运行 `supervisor restart all` 命令即可。

## 5.7 本章小结

本章第一节介绍了系统基本框架搭建，依次介绍了网页端的搭建、网页接口端的搭建和业务逻辑的基础框架搭建；2 到 5 节分别介绍了四个模块的主要界面，接口实现和业务逻辑端服务的实现；第六节介绍了阿里云服务器部署的主要内容和步骤。

## 6 系统测试

### 6.1 测试方法

软件测试是软件开发生命周期阶段的一项重要工作。各种研究分析表明软件测试占了整个软件开发工作的 30%，所以测试对于提高软件质量有至关重要的作用<sup>[11]</sup>。软件测试将程序功能和业务需求结合分析，通过发现并修改软件中的问题以提升软件质量，确保可用性和稳定性。软件测试并不是为了证明这个软件是错误的，而是为了发现软件中的错误，进而结合代码管理、业务需求和相关技术，持续完善软件，通过这种方式，能够尽可能确保软件稳定地持续迭代。常用测试方法有静态测试和动态测试。接下来详细分析一下。

静态测试需要技术人员以手工或自动化的形式对代码逻辑和技术文档进行检查，测试过程不需要运行程序。静态测试中，主要检查代码格式是否符合规范，变量和参数的命名和使用是否有误，是否有逻辑漏洞等，因此静态测试非常耗费时间，而且测试人员需要具备较多的专业知识，对软件开发比较熟悉。因此在实际的测试中，静态测试不经常被使用。

动态测试需要运行程序，使用编写好的测试用例进行测试。动态测试主要有黑盒测试和白盒测试两种测试方法。黑盒测试也叫功能测试，主要针对软件功能和接口进行测试，不需要知道软件内部结构，只需要分析测试结果是否符合预期。白盒测试也称为结构测试，测试人员需要对程序内部结构、功能、逻辑比较熟悉，然后对所有的代码逻辑、代码的正确性进行测试。

开发和测试是分别进行的，测试过程一般由专门的测试人员进行测试。本系统采用的测试方式是黑盒测试。

### 6.2 测试环境

本系统的测试环境如下：

- (1) 部署成功的两台阿里云服务器
- (2) 一台联网的笔记本电脑

### 6.3 功能测试

#### 6.3.1 登录注销模块测试

登录注销模块对登录和注销功能进行了测试，主要对其中的输入、按钮和界面进行了 11 项测试，测试结果全部通过，如表 6.1 所示。

表 6.1 登录注销模块测试表

序号	测试界面	测试项目	操作	预期结果	测试结果
1	登录界面	登录功能	不输入内容，点击“登录”	提示“用户名是必须的”	通过
2	登录界面	登录功能	不输入密码，点击“登录”	提示“密码是必须的”	通过
3	登录界面	登录功能	不输入验证码，点击“登录”	提示“请填写验证码”	通过
4	登录界面	登录功能	用户名输入不存在的“not”，密码输入“password”，验证码输入“aico”，点击“登录”	提示“用户名错误”	通过
5	登录界面	登录功能	用户名输入“admin”，密码输入错误的“password”，验证码输入“aico”点击“登录”	提示“密码错误”	通过
6	登录界面	登录功能	用户名输入“admin”，密码输入错误的“password”，验证码输入“aico”，点击五次“登录”	提示“账户已锁定”	通过
7	登录界面	登录功能	用户名输入“test”，密码输入“password”，验证码输入错误的“verify”，点击“登录”	提示“验证码错误”	通过
8	登录界面	登录功能	用户名输入禁止登录的“disabled”，密码输入“password”，验证码输入“aico”，点击“登录”	提示“禁止登录”	通过
9	登录界面	登录功能	用户名输入“test”，密码输入“1yy40166999”，验证码输入“aico”，点击“登录”	登录成功并跳转到主界面	通过
10	主界面	注销功能	点击用户名下拉框中的退出“登录按钮”	注销成功并跳转到登录界面	通过
11	主界面	注销功能	15 分钟未操作	再次执行其他操作跳转到登录按钮	通过

### 6.3.2 主机资产扫描模块测试

主机资产扫描模块对主机资产监控扫描等功能进行了测试，主要对其中的输入、按钮和界面进行了 13 项测试，测试结果全部通过。如表 6.2 所示。

表 6.2 主机资产扫描模块测试表

序号	测试界面	测试项目	操作	预期结果	测试结果
1	主机资产界面	列表获取功能	点击左侧导航栏“主机资产”或登录成功 在“搜索框”中输入需要搜索的内容 选择点击“复选框”或点击离线主机资产“删除按钮”	显示主机资产列表 显示相应的主机资产列表 弹出确认框，点击确定后删除成功	通过
2	主机资产界面	搜索功能			通过
3	主机资产界面	删除功能			通过
4	主机资产界面	详情功能	点击正在运行的主机资产“详情”按钮	跳转到性能监控界面	通过
5	性能监控界面	数据显示功能	点击主机资产“详情”按钮或点击标签页“性能监控”按钮	显示主磁盘监控、资源监控和资源使用率动态图	通过
6	开放端口扫描界面	列表获取功能	点击标签页“开放端口”扫描按钮	跳转到开放端口扫描界面并显示开放端口列表	通过
7	开放端口扫描界面	详情功能功能	点击“详情”按钮	显示端口详细内容	通过
8	访问控制界面	列表获取功能	点击标签页“访问控制”按钮	跳转到访问控制界面并显示安全组规则列表	通过
9	访问控制界面	新建功能	点击“新建规则”按钮，填写错误的授权对象“aaa”	提示“授权对象不符合格式”	通过
10	访问控制界面	新建功能	点击“新建按钮”，填写正确格式的授权对象	新建安全组规则显示成功	通过
11	基线扫描界面	列表获取	点击标签页“基线扫描”按钮	跳转到基线扫描界面并显示详情和列表	通过
12	基线扫描界面	操作功能	点击“建议”或“忽略”或“恢复”按钮	显示相应的建议详情或扫描结果详情	通过
13	基线扫描界面	开启扫描功能	点击“扫描”按钮	显示最新依次扫描结果详情和列表	通过

### 6.3.3 网络资产扫描模块测试

网络资产扫描模块对新建扫描、扫描结果显示等功能进行了测试，主要对其中的输入、按钮和界面进行了 11 项测试，测试结果全部通过。如表 6.3 所示。

表 6.3 网络资产扫描模块测试表

序号	测试界面	测试项目	操作	预期结果	测试结果
1	网络资产界面	列表获取功能	点击左侧导航栏的“网络资产”	显示扫描项列表	通过
2	网络资产界面	搜索功能	在“搜索框”中输入需要搜索的内容	显示相应的扫描项列表	通过
3	网络资产界面	删除功能	选择点击“复选框”或点击扫描成功或失败操作中的“删除”按钮	弹出确认框，点击确定后删除成功	通过
4	网络资产界面	新建功能	点击“新建扫描”按钮，输入错误格式扫描域“aaa”	提示“新建扫描失败”	通过
5	网络资产界面	新建功能	点击“新建扫描”按钮，输入正确扫描域	新建扫描项在列表中，处于正在扫描状态	通过
6	网络资产界面	详情功能	点击扫描成功的“详情”按钮	跳装到详情界面并显示结果汇总、端口和服务分布图、已扫描 IP 列表	通过
7	详情界面	分布图下载功能	点击分布图“下载图标”	分布图下载到本地	通过
8	详情界面	IP 详情功能	点击已扫描 IP 列表中的“IP”	跳转到 IP 端口服务界面并显示服务列表	通过
9	IP 端口服务界面	服务详情功能	点击“服务详情”按钮	弹出侧边框显示端口服务的详细内容	通过
10	IP 路由追踪界面	链路功能	点击标签页中的“路由追踪”按钮	显示路由追踪路径图并显示追踪依据	通过
11	IP 系统探测界面	探测功能	点击标签页中的“系统探测”按钮	显示系统类型概率饼状图	通过

### 6.3.4 用户管理模块测试

用户管理模块对用户操作进行了 14 项测试，测试结果全部通过。如表 6.4 所示。

表 6.4 用户管理模块测试表

序号	测试界面	测试项目	操作	预期结果	测试结果
1	账户信息界面	数据显示功能	点击左侧导航栏的“账户信息”	显示账户信息	通过
2	账户信息界面	编辑账户功能	点击“账户信息编辑图标”，不输入内容，点击“确定”	提示“不能为空”	通过
3	账户信息界面	编辑账户功能	点击“账户信息编辑图标”，输入内容，点击“确定”	显示修改后的账户信息	通过
4	账户信息界面	修改密码功能	点击“修改密码图标”，输入错误旧密码和新密码，点击“确定”	提示“密码错误”	通过
5	账户信息界面	修改密码功能	点击“修改密码图标”，输入正确内容，点击“确定”	提示“密码修改成功”	通过
6	用户管理界面	列表获取功能	点击左侧导航栏的“用户管理”	跳转到用户管理界面并显示用户列表	通过
7	用户管理界面	搜索功能	在“搜索框”内输入搜索内容后敲击“回车”	显示相应的用户列表	通过
8	用户管理界面	数据过滤功能	选择点击“登录状态”和“用户角色”下拉框	显示相应的用户列表	通过
9	用户管理界面	删除功能	选择点击“复选框”后点击“批量删除”按钮或点击操作中的“删除图标”	删除的用户不再出现在用户列表中（管理员账户不可删除）	通过
10	用户管理界面	添加用户功能	点击“添加用户”按钮后在表单不输入内容，点击确定	提示“不能为空”	通过
11	用户管理界面	添加用户功能	点击“添加用户”按钮后在表单输入内容，点击确定	提示“用户添加成功”	通过
12	用户管理界面	编辑用户功能	点击“账户信息编辑图标”，输入内容，点击“确定”	提示“编辑成功”	通过
13	用户管理界面	重置用户密码功能	点击“重置密码图标”，点击“确定”	提示“重置成功”	通过
14	用户管理界面	修改登录状态功能	点击除管理员的其他用户的“登录开关”	开关切换	通过

## 6.4 测试结果与分析

本系统使用了黑盒测试当时对四个模块的各个功能进行了测试。在登录注销模块中，测试了输入的合法性和不合法性，也对安全控制，比如账户锁定和是否允许登录进行了测试。在主机资产扫描模块中，主要验证了需求中的各部分数据显示和访问控制的安全组规则创建进行了测试。网络资产扫描模块主要测试了新建扫描的数据的合法性、以及界面跳转和数据展示。在用户管理模块中，首先测试了账户信息获取、编辑和密码修改，然后对用户管理的增删改查功能进行了测试。

在测试过程中也发现了部分漏洞，比如提示信息不够全面，数据展示不全面，接口逻辑漏洞等，通过在本地环境测试修改然后重新部署对这些漏洞进行了修复，而后再次进行了测试，最终，所有的测试用例全部通过。

## 6.5 本章小结

本章第一节介绍了常用的测试理论知识，第二节对测试环境进行了简要介绍，第三节对各个模块的测试内容进行了详细的说明，第四节对测试结果进行了总结与分析。最终所有的测试全部通过，说明系统功能实现符合需求预期。

## 结 论

基于 **ZMap** 的网络资产扫描系统旨在开发一款即可以对网络资产进行扫描也可以对主机资产进行安全管理的系统，该系统以网页的形式进行数据展示和交互，方便用户使用，另外还提供了用户权限的管理机制，在支持网络资产扫描的同时，也十分方便多人对中小型的主机系统网络进行管理。

网络资产扫描系统的开发分为网页端与服务器端两部分。服务器端又可拆分为网页接口端和业务逻辑端，方便进行管理和开发。网页端开发使用了 **Ant Design** 和 **UmiJS** 框架，开发语言是 **TypeScript**。网页接口端开发主要使用了 **IRIS**、**gRPC** 框架，另外也使用了 **JWT** 对进行用户鉴权，业务逻辑端开发使用了 **gRPC** 框架，开发语言主要是 **Golang**，最终这些服务都部署到了阿里云服务器上。

网络资产扫描系统主要分为四个模块。对于登录注销模块，具有登录权限的用户可以进行登录，用户登录成功之后网页接口端会生成一个 **JWT** 返回给网页端，在 **JWT** 有效期内或用户未注销时用户再次进入系统不需要再次登录。在主机资产扫描模块中，只有管理员可以对注册到系统的主机资产进行管理，包括性能的监控和安全控制两部分。在网络资产扫描模块中，所有的登录用户都可以发起网络资产扫描，并且查看扫描的详细内容。用户管理模块中除了账户信息外，对用户的管理只有管理员才可以操作，方便管理员对所有用户进行统一管理。

网络资产系统也有一定的不足之处，比如在针对存活主机较多的 **CIDR** 进行网络资产扫描时速度会比较慢，没有对主机资产的代理程序做注册时权限限制，界面数据展示不够美观，之后可以在这几个方面入手，增强系统性能和安全性，提高用户体验。

## 参 考 文 献

- [1] 王宸东, 郭渊博, 甄帅辉, 杨威超. 网络资产探测技术研究[J]. 战略支援部队信息工程大学, 2018.
- [2] 陈健, 钱星桥. 网络资产识别与安全风险探测软件开发[J]. 技术与市场, 2021.
- [3] 张星亮, 张洪波. 网络安全扫描工具的比较与分析[J]. 电脑知识与技术: 学术版, 2018(4Z): 2.
- [4] Shah M, Ahmed S, Saeed K, et al. Penetration testing active reconnaissance phase optimized port scanning with nmap tool[C]. 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies. IEEE, 2019: 1–6.
- [5] Mazel J, Strullu R . Identifying and characterizing ZMap scans: a cryptanalytic approach[J]. 2019.
- [6] Lee S, Im S, Shin S H, et al. Implementation and vulnerability test of stealth port scanning attacks using ZMap of censys engine[C]. International Conference on Information and Communication Technology Convergence. IEEE, 2016: 681–683.
- [7] Bennett C, Abdou A R, Oorschot P. Empirical Scanning Analysis of Censys and Shodan[C]. Workshop on Measurements, Attacks, and Defenses for the Web. 2021.
- [8] 冯志勇, 徐砚伟, 薛霄, 等. 微服务技术发展的现状与展望[J]. 计算机研究与发展, 2020, 57(5): 20.
- [9] Himschoot P. Efficient Communication with gRPC[M]. Microsoft Blazor. Apress, Berkeley, CA, 2022: 465–482.
- [10] Wan G, Izhikevich L, Adrian D, et al. On the origin of scanning: The impact of location on Internet-wide scans[C]. Proceedings of the ACM Internet Measurement Conference, 2020: 662–679.
- [11] 张新, 焦豪. 两种黑盒测试方法的比较分析[J]. 电子技术与软件工程, 2018(7): 3.
- [12] 陈洋, 罗敏, 常珞. 基于 etcd 集群 https 通信的实现[J]. 电子技术与软件工程, 2020.
- [13] Kropp A , Torre R . Docker: containerize your application[J]. Computing in Communication Networks, 2021.
- [14] Chai X C , Wang Q L , Chen W S , et al. Research on a Distributed Processing Model Based on Kafka for Large-Scale Seismic Waveform Data[J]. IEEE Access, 2020, PP(99): 1–1.
- [15] Liu W L , Ltd S , China. Design of Multiple Environments CI/CD Solution[J]. Modern Computer, 2019.
- [16] 李若兰. 基于 Nginx 的 Web 服务器优化的应用研究[J]. 科技风, 2021.
- [17] 陈宇收, 饶宏博, 王英明, 等. 基于 JWT 的前后端分离程序设计研究[J]. 电脑编程技巧与维护, 2019(9): 2.

- [18] Singh R K , Verma H K . Redis Based Messaging Queue and Cache Enabled Parallel Processing Social Media Analytics Framework[J]. Journal of King Saud University-Computer and Information Sciences, 2020.
- [19] 代飞, 刘国志, 李章, 等. 微服务技术:体系结构, 通信和挑战[J]. 应用科学学报, 2020, 38(5):18.
- [20] 赵瑜颤. 基于 Grpc 的分布式远程过程调用框架设计与开发[J]. 现代信息科技, 2021.

## 修改记录

### (1) 毕业设计（论文）内容重要修改记录

#### 第一次修改记录：

修改前：章节名称含义不清楚，体现不出测试方法。

修改后：将第六章测试理论更改为测试方法。

#### 第二次修改记录：

修改前：系统实现部分缺少相应的结构图。

修改后：将结构图进行了补充，并加以详细的描述。

#### 第三次修改记录：

修改前：参考文献格式错误，内容缺失。

修改后：参照论文模板进行规范，修改了缩进、对齐格式，补充了期刊的期号、卷号、页码

#### 第四次修改记录：

修改前：目录编号错误和图片留白过多，编号缺失。

修改后：参照论文模板进行规范，增加了编号并修改图片展示。

### (6) 毕业设计（论文）正式检测重复比：2.8%

记录人（签字）：冷阳煜

指导教师（签字）：丁静

## 致 谢

论文的撰写已接近尾声，这也就预示着我的大学生活即将结束。在这里，我要向我的家人、老师和朋友表示最真诚的感谢。

首先我要感谢我的家人对我的无微不至。每当我犯错时，我的家人总能以最包容的心态来给予我物质和精神上的支持，并鼓励我勇往直前。其次，我要感谢我的辅导员祝晓璇祝老师，她总是能够时刻关注我的生活，在这几年的大学生活中，她的出现灿烂无比，在生活、学业和心理上都给予我巨大的帮助。还要感谢我的指导老师丁锋老师，丁老师在选题和开题上都为我花费了巨大的精力，并在后续实践中为我提供了建议和帮助。最后，感谢我曾经的同事和身边的同学。光阴飞逝，前程犹在，祝大家前程似锦，一马平川！