**北京外国语大学网络教育学院**

经管专业毕业设计（论文）

CMDB资产管理系统的设计与实现

姓 名： \_\_\_\_\_\_\_\_\_\_\_\_\_\_魏泽民\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学 号： \_\_\_\_\_\_\_\_\_\_\_\_132611010021\_\_\_\_\_\_\_\_\_\_\_\_\_\_

专 业： \_\_\_\_\_\_\_信息管理与信息技术\_\_\_\_\_\_\_\_\_\_\_

指导老师： \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

论文提交日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**北京外国语大学网络教育学院**

**学士学位论文诚信声明**

本人郑重声明： 所呈交的学士学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。论文所涉及的项目为本人亲自负责或者参与实施的项目。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。本人完全了解本声明的法律结果由本人承担。

学士学位论文作者签名:

日期： 年 月 日

目 录

[1 绪 论 6](#_Toc499681834)

[1.1 课题研究的背景与意义 6](#_Toc499681835)

[1.2 CMDB介绍 6](#_Toc499681836)

[1.2.1 事件管理（Incident Management） 6](#_Toc499681837)

[1.2.2 问题管理（Problem Management） 6](#_Toc499681838)

[1.2.3 配置管理（Configuration Management） 6](#_Toc499681839)

[1.2.4 变更管理（Change Management） 6](#_Toc499681840)

[1.2.5 发布管理（Release Management） 7](#_Toc499681841)

[2 设计综述 8](#_Toc499681842)

[2.1 需求分析 8](#_Toc499681843)

[2.2 开发环境 8](#_Toc499681844)

[2.3 项目结构设计 8](#_Toc499681845)

[2.4 数据库设计 9](#_Toc499681846)

[3 功能实现思路 12](#_Toc499681847)

[3.1 登录实现 12](#_Toc499681848)

[3.2 注册实现 13](#_Toc499681849)

[3.3 资产信息展示 14](#_Toc499681850)

[3.4 资产信息添加 15](#_Toc499681851)

[3.5 资产信息修改 15](#_Toc499681852)

[3.6 IDC及业务线信息添加实现 16](#_Toc499681853)

**摘 要**

随着互联网不断的发展，越来越多的企业将应用系统构架在PC、LAN、服务器、防火墙等互联网设备上，IT系统发展至今已经成为业务系统不可或缺的支撑基础。公司机构的 IT 资源正在不断扩大，如今，IT 部门负责人必须监控和管理服务器、云服务、软件、移动设备、网络架构以及令人眼花缭乱的终端设备：从台式机、笔记本到平板电脑、智能手机和智能手表。随着ITIL管理标准的提出，资产管理系统应运而生。

CMDB是**Configuration Management Database（配置管理数据库）的简称。CMDB 存储与管理企业 IT 架构中设备的各种配置信息，它与所有服务支持和服务交付流程都紧密相联，支持这些流程的运转、发挥配置信息的价值，同时依赖于相关流程保证数据的准确性。**

**本系统使用python语言开发，使用Django web框架，注意使其符合操作的业务流程，力求呈现界面友好、提高企业资产管理的水平和效率。**

**。。。。。。**

**关键词：python；Django；配置信息；管理；数据库**

**Abstract**

With the further development of the Internet, more and more enterprises will be of application architecture in PC, LAN, server, firewall and other Internet devices, IT system development has become an indispensable business based support system. The company's IT resources are expanding, now, the terminal equipment IT departments must monitor and management server, cloud services, software, mobile devices, network architecture and dazzling: from desktop and laptop to tablet computer, intelligent mobile phone and smart watch. With the introduction of ITIL management standards, CMDB came into being.

CMDB is the abbreviation of Configuration Management Database (configuration management database). All kinds of equipment configuration information storage and management of CMDB enterprise IT architecture, with all service support and service delivery processes are closely related to the value support of these processes running, play configuration information, at the same time depends on the process to ensure the accuracy of the data.

In the process of development, the system should pay attention to make it conform to the business process of operation, and strive to show friendly interface and improve the level and efficiency of enterprise asset management.

。。。。。。

# 绪 论

## 课题研究的背景与意义

互联网迅猛发展的时代，越来越多的企业将业务与互联网结合，企业内部的管理也随之发生变化，IT系统发展至今已成为企业管理架构中不可或缺的支撑基础。每个企业都会通过OA或者各种相似的管理平台来实现现代化管理。对于企业的基础资产传统的管理方式也有相应的新办法，ITIL管理标准的提出，资产管理系统（CMDB）便应运而生。

## CMDB介绍

CMDB——Configuration Management Database 配置管理数据库，CMDB存储与管理企业IT架构中资产的各种配置信息，它与所有服务支持和服务交付流程都紧密相联，支持这些业务流程的运转、发挥配置信息的价值，同时保证相关服务获得数据的准确性和一致性。

CMDB至少实现以下几个方面的管理能力：

### 事件管理（Incident Management）

事故管理负责记录、归类和安排专家处理事故并监督整个处理过程直至事故得到解决和终止。事故管理的目的是在尽可能最小地影响客户和用户业务的情况下使IT系统恢复到服务级别协议所定义的服务级别。

### 问题管理（Problem Management）

问题管理是指通过调查和分析IT基础架构的薄弱环节、查明事故产生的潜在原因，并制定解决事故的方案和防止事故再次发生的措施，将由于问题和事故对业务产生的负面影响减小到最低的服务管理流程。与事故管理强调事故恢复的速度不同，问题管理强调的是找出事故产生的根源，从而制定恰当的解决方案或防止其再次发生的预防措施。

### 配置管理（Configuration Management）

配置管理是识别和确认系统的配置项，记录和报告配置项状态和变更请求，检验配置项的正确性和完整性等活动构成的过程，其目的是提供IT基础架构的逻辑模型，支持其它服务管理流程特别是变更管理和发布管理的运作。

### 变更管理（Change Management）

变更管理是指为在最短的中断时间内完成基础架构或服务的任一方面的变更而对其进行控制的服务管理流程。变更管理的目标是确保在变更实施过程中使用标准的方法和步骤，尽快地实施变更，以将由变更所导致的业务中断对业务的影响减小到最低。

### 发布管理（Release Management）

发布管理是指对经过测试后导入实际应用的新增或修改后的配置项数据，进行分发和宣传的管理流程。

# 设计综述

## 需求分析

现针对互联网企业IT资产管理普遍需求情况，根据CMDB可实现的管理能力，设计并开发适用的管理平台。功能需求如下：

1. 存储所有的IT资产信息，合理的分表结构
2. 信息可视化，拥有良好的交互界面
3. 按照业务服务信息分类
4. 可靠的数据添加、修改和删除
5. 良好的扩展能力，为对接其他管理系统做准备

## 开发环境

前端框架：Bootstrap

前端开发语音：JS、CSS、Jquery、HTML

后台开发语音：Python3

Web框架：Django

数据库： Mysql

## 项目结构设计

使用Django框架MTV架构设计，目的是为了各组件之间保持松耦合关系。Django的MTV分别是指：

M代表模型（model）：负责业务对象和数据库的关系映射（ORM）

T代表模板（Template）：负责如何把页面展示给用户（html）

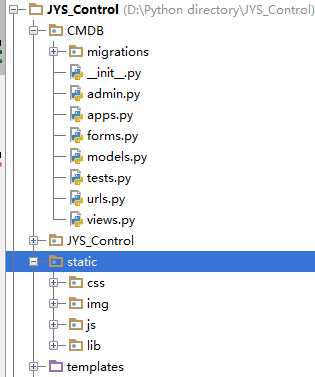
V代表视图（View）：负责业务逻辑，并在适当的时候调用Model和Template

除了以上三层之外，还有一个URL分发器，它的作用是将一个个URL的页面请求分发给不同的View处理，View再调用对应的Model和Template。

在本项目开发中将项目以APP的方式添加在Django中，APP中的models.py文件定义该APP的数据库模型。

并增加static目录，统一将JS、CSS和图片等静态资源收集管理

最终结构如下图：



## 数据库设计

用户、分组、权限关联均使用Django框架自动生成，表结构如下

|  |  |
| --- | --- |
| **用户表** | |
| Username | 用户名 |
| First\_name | 名字 |
| Last\_name | 姓 |
| Password | 密码 |
| Last\_login | 上一次登录时间 |
| Is\_superuser | 是否为超级用户 |
| Email | 邮箱 |
| Is\_staff | 权限级别 |
| Is\_active | 禁用状态 |
| Date\_joined | 创建时间 |

|  |  |
| --- | --- |
| **用户分组表** | |
| Name | 组名 |

|  |  |
| --- | --- |
| **用户和组关联表** | |
| User\_id | 用户id |
| Group\_id | 组id |

|  |  |
| --- | --- |
| **权限表** | |
| Name | 权限名称 |
| Content\_type\_id | 权限级别编号 |
| Codename | 权限别名 |

|  |  |
| --- | --- |
| **组权限关联表** | |
| Group\_id | 组id |
| Permission\_id | 权限id |

|  |  |
| --- | --- |
| **用户权限关联表** | |
| User\_id | 用户id |
| Permission\_id | 权限id |

资产表、业务线、数据中心、服务器信息表需要单独创建，表结构如下

|  |  |
| --- | --- |
| **资产表** | |
| Device\_type\_id | 设备类型编号 |
| Device\_status\_id | 设备状态编号 |
| Cabinet\_num | 机柜号 |
| Cabinet\_order | 机柜中序号 |
| Idc | 机房 |
| Business\_unit | 所属业务线 |

|  |  |
| --- | --- |
| 业务线表 | |
| Name | 业务名称 |

|  |  |
| --- | --- |
| 数据中心表 | |
| Name | 数据中心名称 |
| Addres | 地址 |
| Telephone | 联系电话 |

|  |  |
| --- | --- |
| 服务器信息表 | |
| Ip | IP地址 |
| Model | 型号 |
| Manufacture | 品牌 |
| SN | SN码 |
| Os\_platform | 系统平台 |
| Os\_version | 系统版本 |
| CPU | Cpu型号 |
| CPU\_physical\_count | CPU数量 |
| CPU\_count | CPU核心数量 |
| Mem | 内存型号 |
| Mem\_val | 内存容量 |
| Create\_at | 创建时间 |
| Asset\_id | 关联的资产编号 |

# 功能实现思路

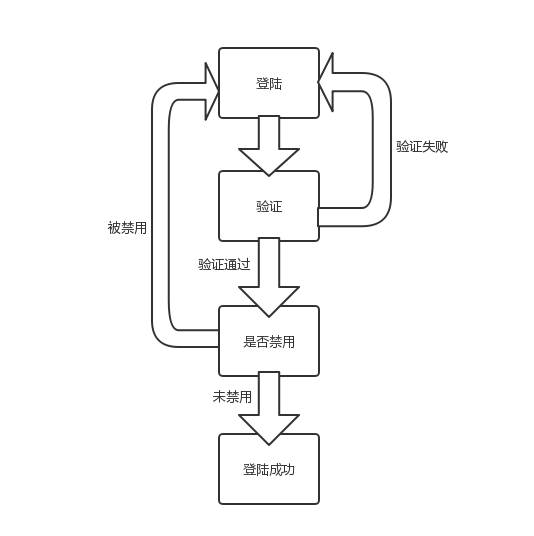
## 登录实现

登录页面只进行了简单设计，主要就是用户名和密码的输入框和登录提交按钮，例如：



使用AJAX将用户名、密码和CSRF\_TOKEN值提交至后台，此处加入了CSRF防跨站攻击和session判断相结合，可在接收请求的时候判断用户的真实性，防止被篡改。

登陆流程图如下：



## 注册实现

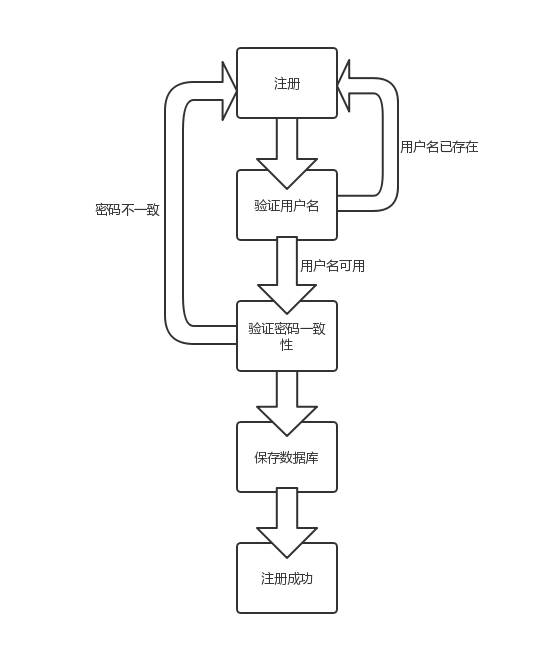
注册页面：



注册输入验证错误信息提示：



注册流程图：



## 资产信息展示

资产信息页面设计效果如图：



后台首先从数据库资产信息表、服务器信息表和业务线表查询已经录入的信息，获得有效信息后，分别提取并组合成字典格式，使用render方法渲染并在前端页面利用DOM元素展示。

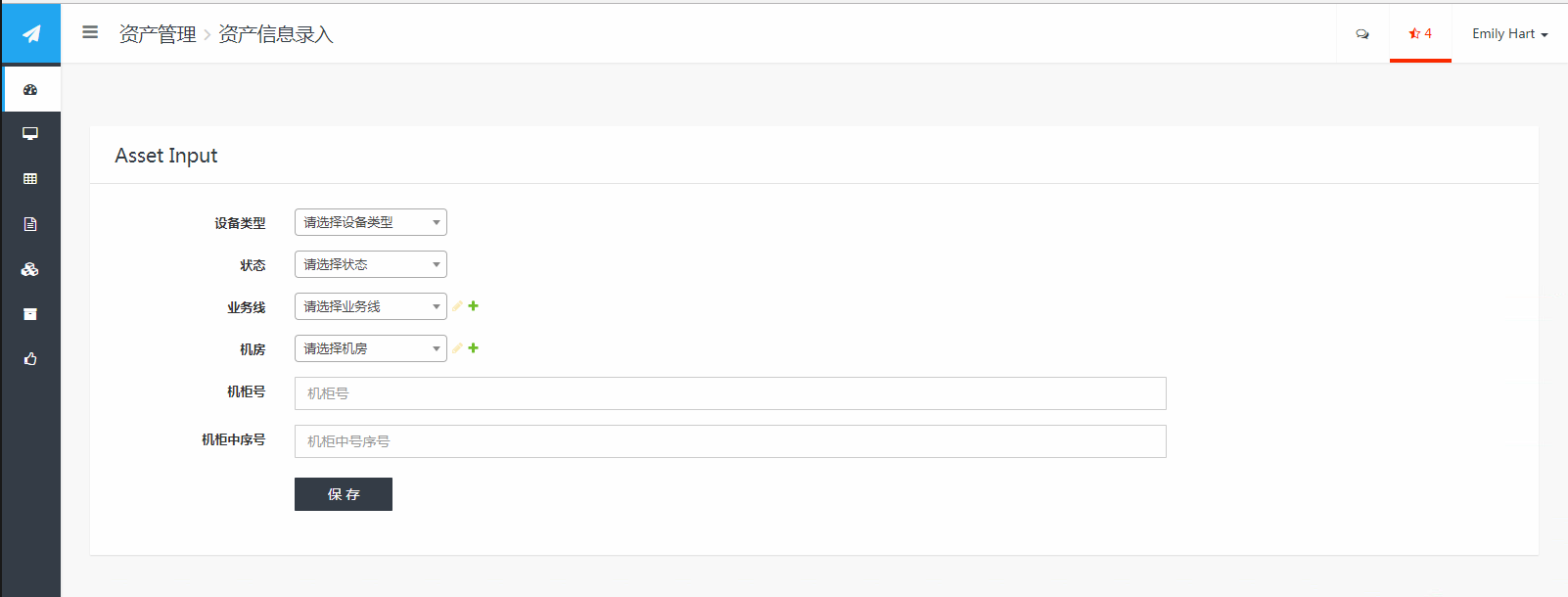
页面加入了资产添加按钮。

资产编号为机房名称、机柜号、机柜中序号拼接生成，点击资产编号可以跳转至对应的资产修改页面。

## 资产信息添加

通过点击资产信息页面中的增加资产信息按钮即可跳转至添加页面。

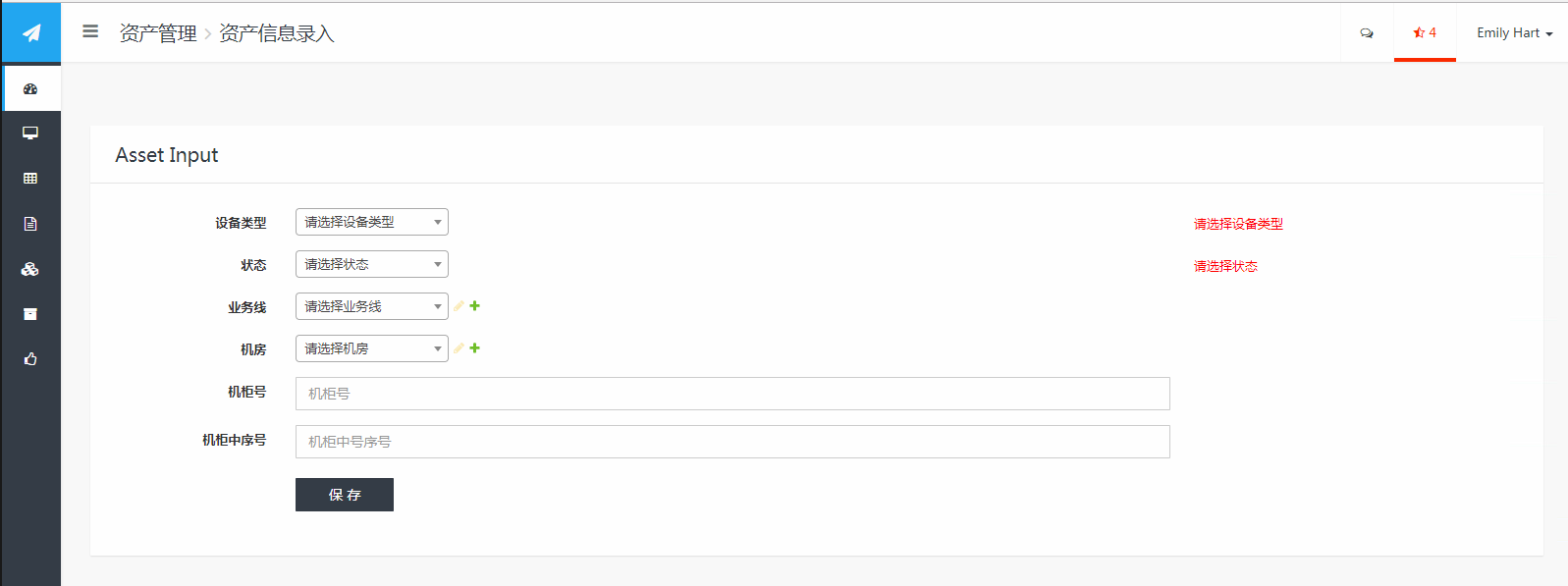
资产添加页面设计效果如下：



该页面中下拉框中的选项是根据数据库中的数据实时动态生成，业务线、机房可以实现实时添加、修改和删除功能，利用localstorage功能跨页面传递参数，从而实时动态的展示修改后的数据。

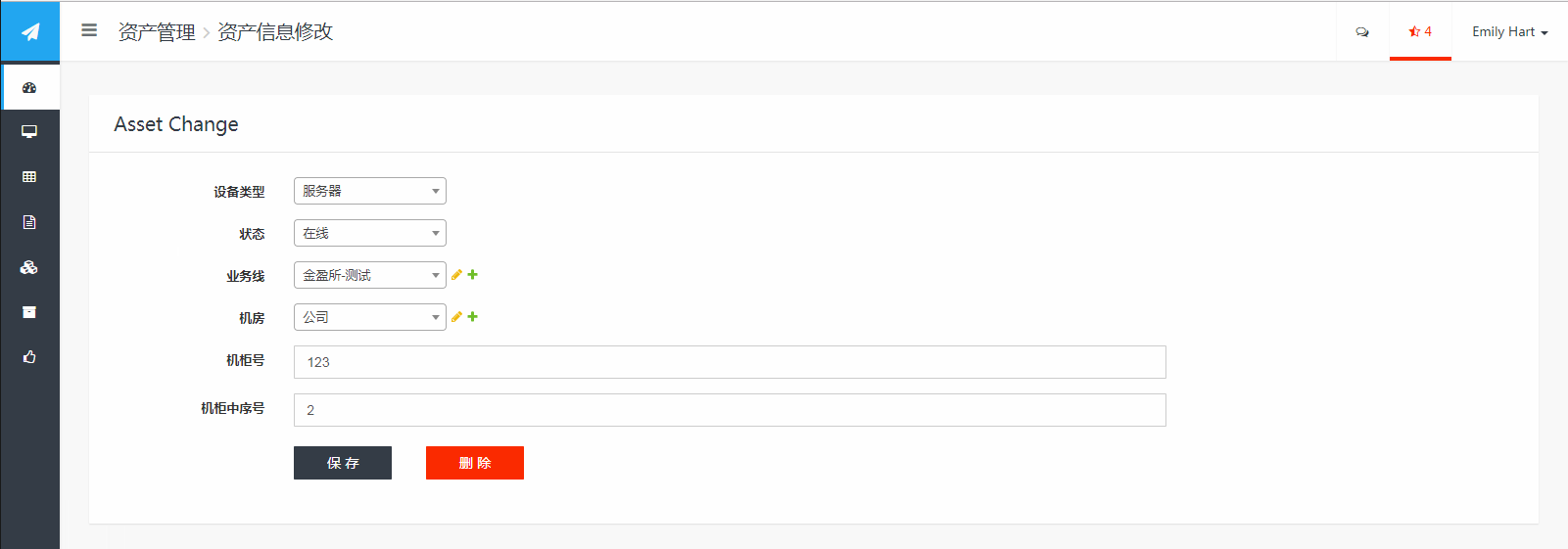
修改按钮仅在选择了选项之后才为可点击状态，若未选择则无法点击，增加用户体验以及防止错误的修改操作。

保存数据时，后台会通过forms.py文件中的自己定义的验证规则，验证输入值的合规性，若通过则保存至数据库。否则返回错误信息。如图：



## 资产信息修改

在资产信息展示页面点击资产编号即可跳转修改页面，页面设计效果如图：

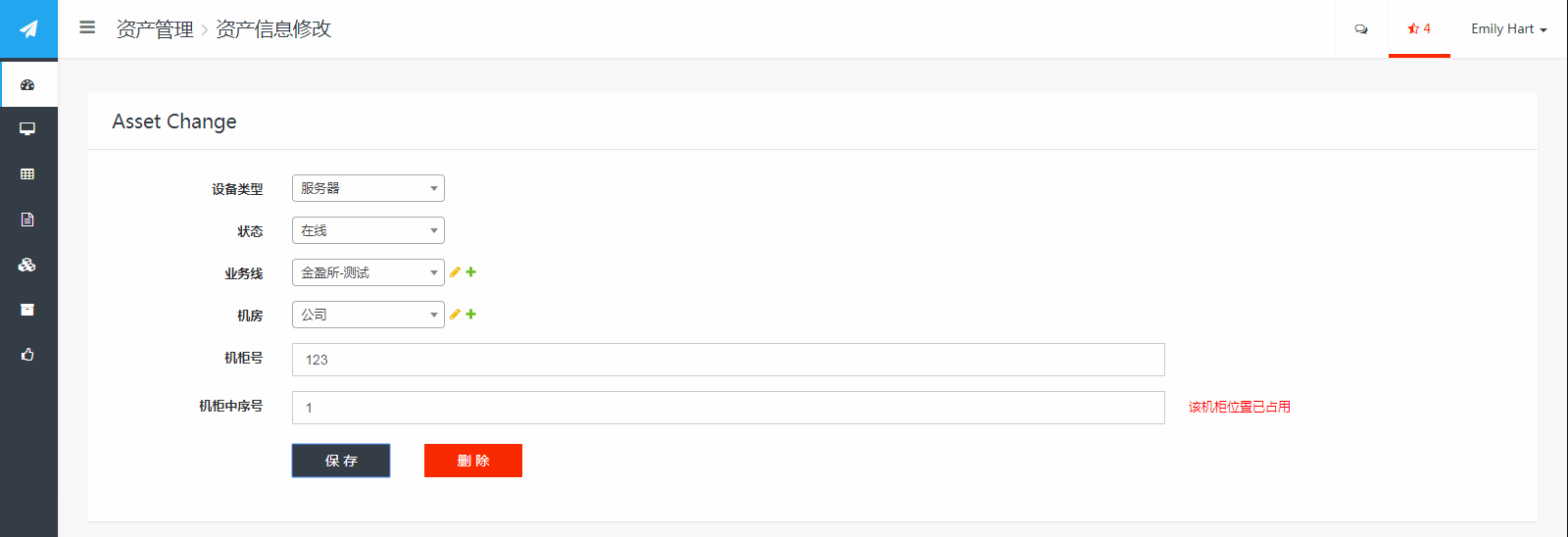


修改页面能获取数据库对应的数据关键一点是，前端页面在URL中拼接资产id参数，如：

href="/data\_input/asset\_change?asset\_id={{ item.id }}"

后台views中的方法获得资产id之后，从数据库中查询出相关数据，然后返回给前端页面展示。

后台保存修改数据的时候依然会验证数据的合规性，并且会根据机柜和机柜中序号判断是否与其他资产信息冲突，如果冲突则提示错误信息。如图：



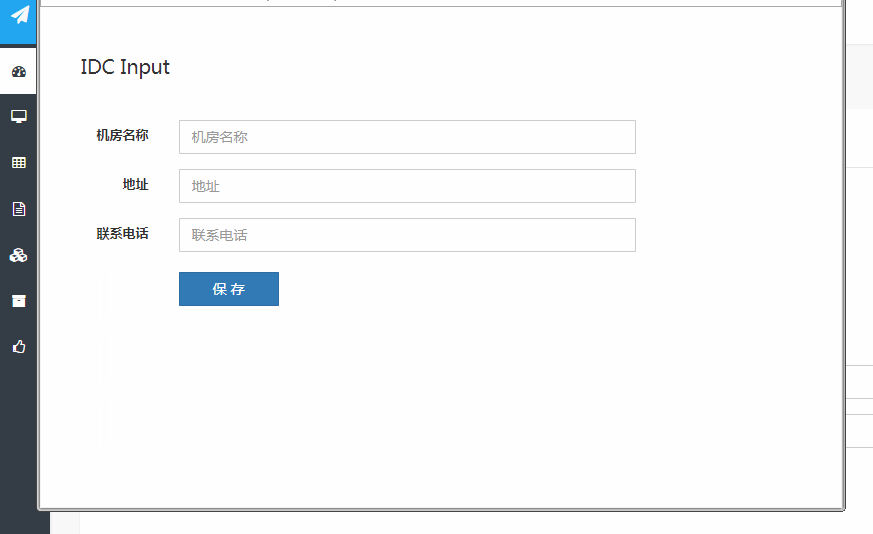
删除方法则简单直接，后台根据前端返回的资产信息id，删除数据库中对应id的资产信息数据，并返回到资产信息展示页面。

## IDC及业务线信息添加实现

在资产信息添加及修改页面中，都包含了机房和业务线的添加、修改和删除功能。如图：

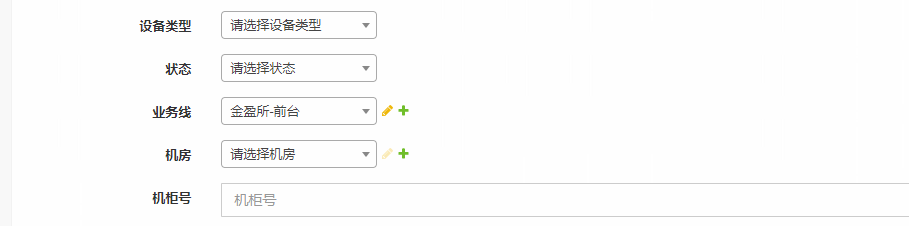


两个模块实现方式相同，通过点击加号弹出添加页面。如图：

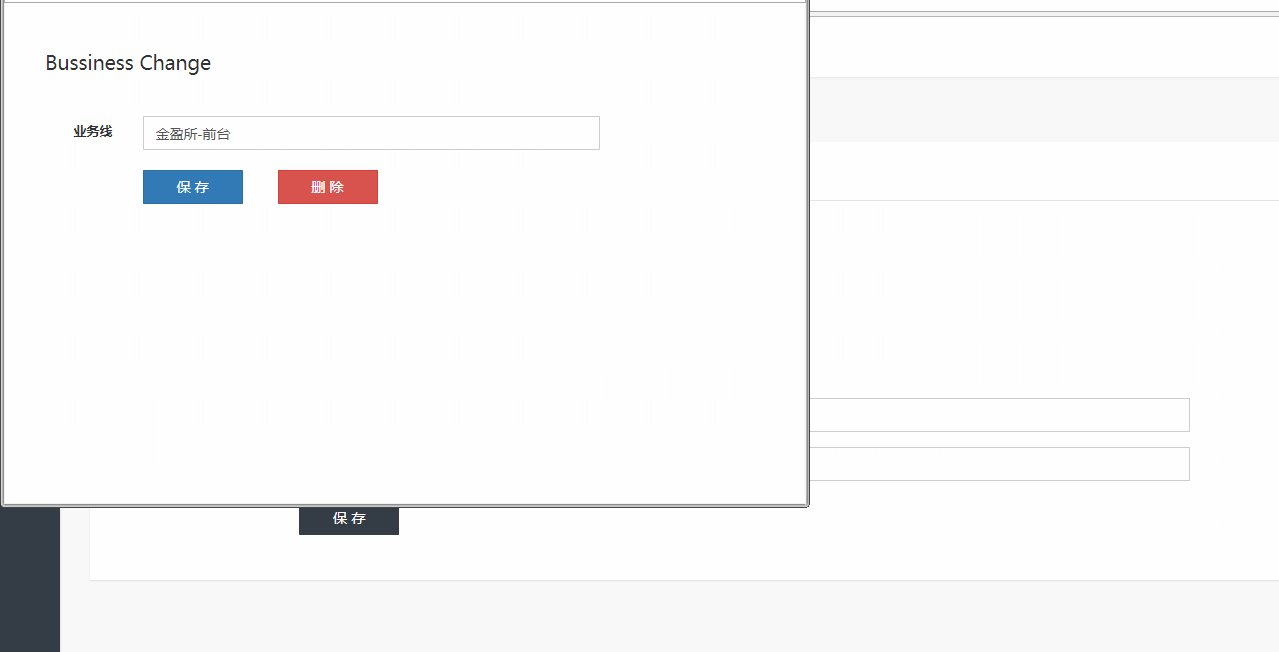


此处利用localstorage监听实现了跨页面传值，从而可以动态的修改前一个页面下拉框的内容，比如此时添加一个新的IDC信息，前一个页面不需要刷新，下拉框中即可选择刚刚添加的选项。

修改部分的前段则要多一些判断，根据下拉框的值来改变按钮的样式和可点击状态。此处使用JS实现，效果如图：



点击铅笔状修改按钮弹出修改页面。如图：



此处修改效果一样是可以立即展示在前一个页面，而避免了刷新页面的操作。所有修改和删除的请求通过AJAX的POST方法，提交至后台Views.py文件中对应的方法来处理。

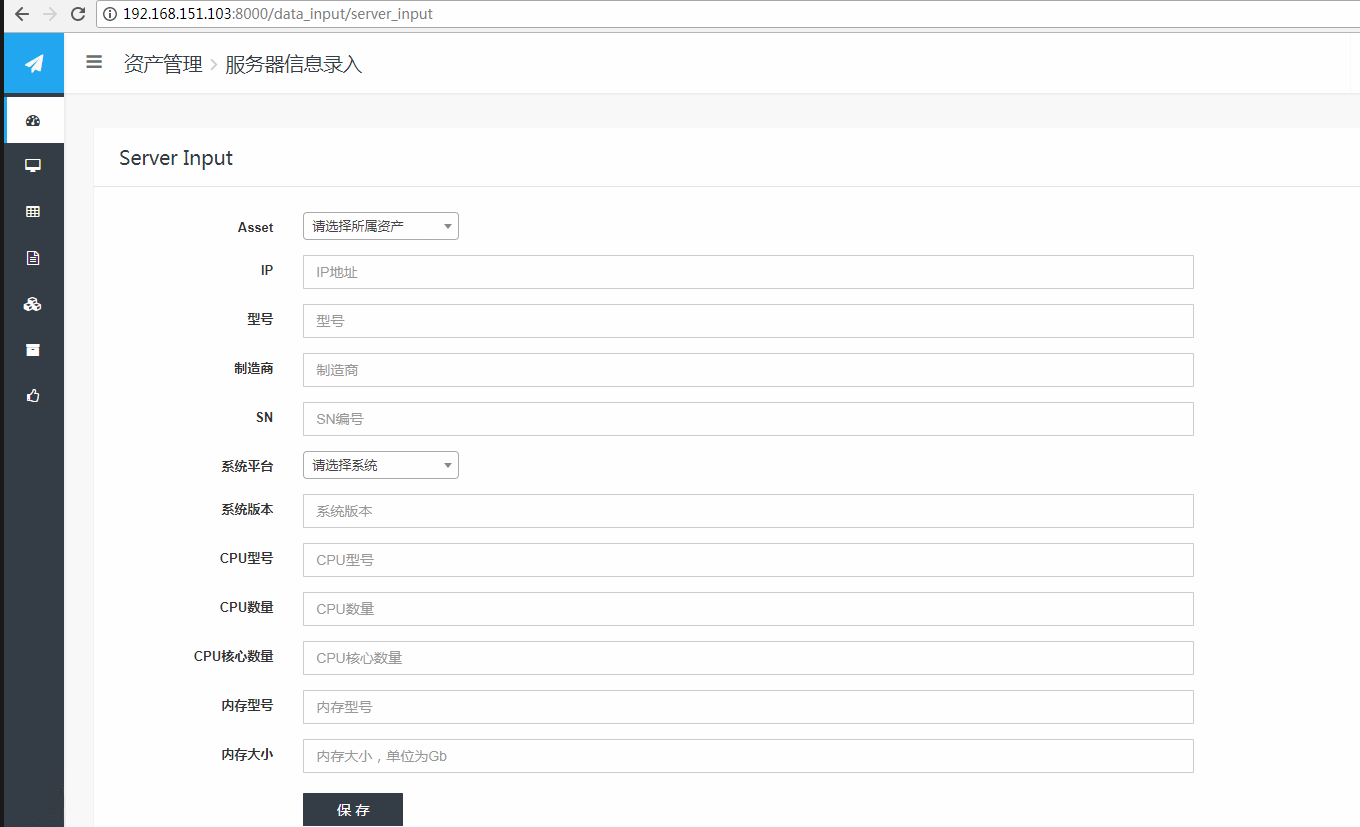
## 服务器信息展示、添加和修改实现

服务器展示页面效果如图：

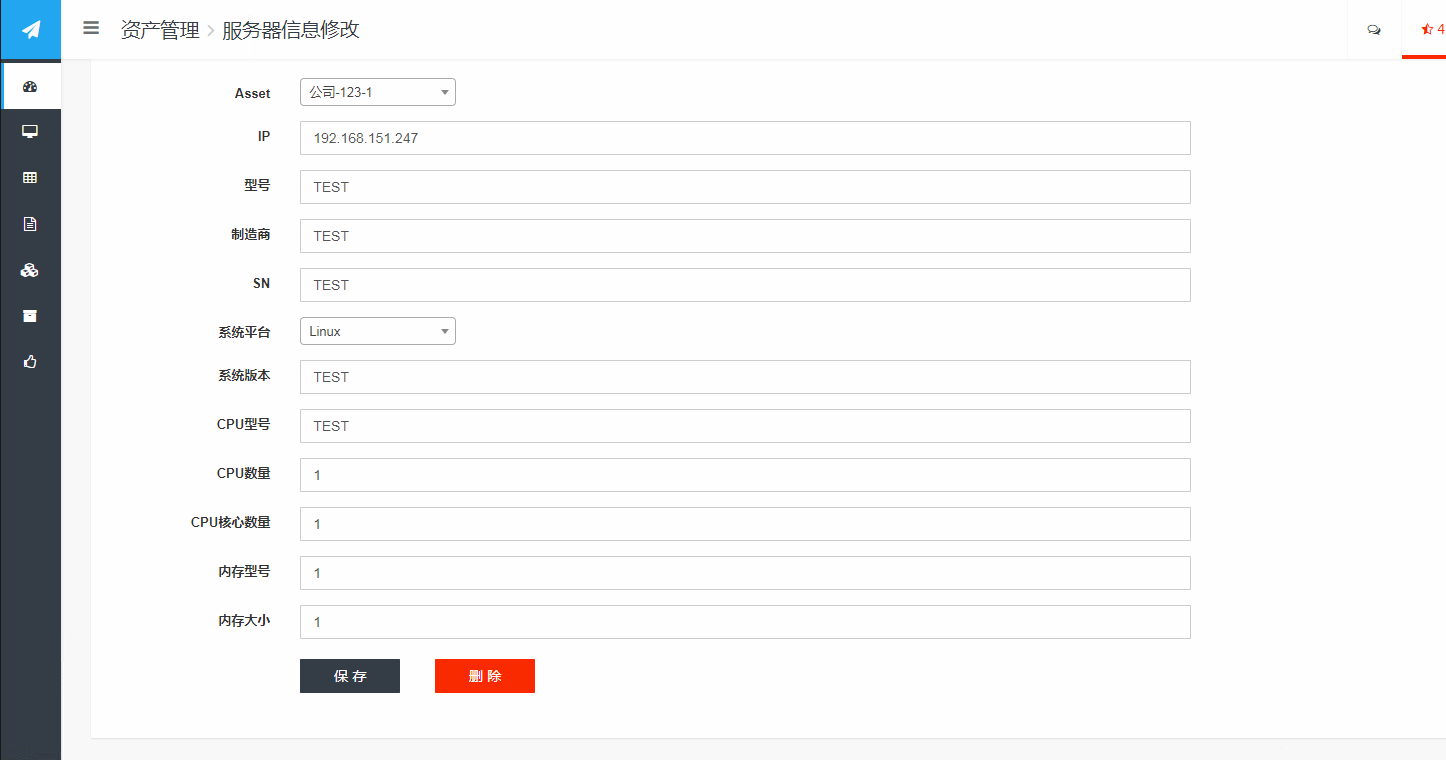


服务器展示页面独立于资产信息，并存在于主导航栏中，与资产信息逻辑上关联，在页面展示功能上与资产信息无太大差别。

但由于在数据库模型Model设计中服务器信息与资产信息为OneToOne的关系，所以在操作流程上要先添加资产信息，然后在添加服务器信息时选择相应的资产信息并关联起来。保存的时候所有输入的数据都会在forms.py对应的方法里进行验证，如果合规则保存在对应的数据库中。如图：



同样点击IP则可以跳转至该条信息对应的修改页面，后台通过前端返回的服务器id查询出对应数据并返回给页面。如图：



同样在保存的时候有form验证数据的合规性，不合格则返回错误信息。

## 分页功能实现

在程序优化和数据库优化角度，当数据量相当多的时候，一次性查询出所有数据将是特别耗费服务器资源的，可能会造成系统和页面发生未响应的情况。所以通常在页面展示数据的时候，通常对数据做分页处理，即显示页面一次只展示有限的数据，通过翻页来部分读取，从而避免对数据库查询造成过大压力。

分页功能可以通过修改数据库查询规则来实现，在python中也有专门处理分页数据的模块。本着面向对象编码的原则，避免对每个数据展示的方法都添加分页一个完整的方法，这里使用Paginator模块实现分页功能，并将该功能单独写成一个类方法，从而减少各个后台方法的代码复杂度，进一步实现模块化功能。

当一个方法要使用分页功能时，只需要将数据对象传入分页的类方法中，便会计算出最佳的分页页数、分页标签个数等，获得处理后的分页对象，再返回前端页面进行渲染即可。

前端页面会根据分页对象中的数据，自动判断并改变样式，例如自动判断是否显示上一页、下一页、首页和尾页的标签按钮。

完成后的分页效果如图：



点击页码动态更新表格中数据内容，并且判断页码来改变，页码按钮框的样式和数值。

## 强制登录功能实现

资产管理系统保存了企业的资产信息以及相关互联网设备的配置信息，从信息安全方面考虑，应该限制为仅登录用户可以查看，现在虽然有登录功能，但是用户直接访问资产信息展示页面的URL，是可以看到页面内容的。

那现在就有了一个新需求：

1. 用户登录系统才可以访问某些页面
2. 如果用户没有登录而直接访问就会跳转到登录页面

在Django框架中，[要实现这样的需求通常的办法是使用@login\_required装饰器，但这样做有个问题，就是要在Views.py](mailto:要实现这样的需求通常的办法是使用@login_required装饰器，但这样做有个问题，就是要在Views.py)文件中所有需要强制登录的方法前面都要加上这个装饰器，这样操作比较繁琐，而且一旦修改起来十分麻烦。

在这里我们尝试使用了新的方法来实现需求，建立middleware.py文件，并加入到Django框架的中间件调用中，使用中间件来拦截浏览器请求，然后获取浏览器cookie中的sessionid和数据库的session表中的值做比对，若session存在并且未到过期时间，则认为用户已经登录，否则跳转回登录页面。

使用中间件的好处，不影响程序的主逻辑代码；避免繁琐的在各个方法前加装饰器；在请求拦截中触发，精确控制；只需要一个方法即可，针对不同需求的时候，可以轻松修改，适用性比较强。