南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论文题目	基于 Flask 框架的物资管理系统的设计与实现
作者姓名	吴 桐
学科、专业名称	工程硕士(软件工程方向)
研究方向	软件工程
指导教师	任桐炜　副教授

2016 年 4 月 15 日

学　　号：**MF1432070**

论文答辩日期：　　**2016** 年　**5** 月 **18** 日

指 导 教 师：　　（签字）

基于 Flask 框架的物资管理系统的设计与实现

作　　者: 吴桐

指导教师: 任桐炜　副教授

南京大学研究生毕业论文

(申请工程硕士学位)

南京大学软件学院

2016 年 5 月

The Design and Implementation of Inventory Management System Based on Flask framework

Wu, Tong

Submitted in partial fulfillment of the requirements for the degree of Master of Engineering

Supervised by

Professor **Ren, Tongwei**

Software Institute

NANJING UNIVERSITY

Nanjing, China

May, 2016

摘 要

目前大部分企业在拥有一批物资需要进行跟踪、记录和管理时，通常可能采用最传统的管理方式，即通过 Excel 增删改记录来管理。但是这样的方式存在着许多弊端，显得繁琐、易出错。因此，对于现代企业来说，当物料资源在公司的有形财产中占了比较大的份额时，物资管理就变得更加重要，这时就需要一个内部管理平台来对这些物资进行良好的管理。

本文所设计与实现的物资管理系统，是一个应用于公司内部的网上管理系统；是着力于改善传统的 Excel 表格记录和管理的方式，实现物资管理自动化的信息系统；是能够对物资进行资源请求、归还、报废、转交、编辑、检索等操作，能够导入导出 Excel 表格，进行在线消息通知和邮件通知，让物资的状态变化及时通知到相关人员的管理信息系统。

系统采用了三层结构，表现层使用了 HTML、CSS、JavaScript、JQuery、Bootstrap 框架技术以及 jinja2 模板引擎，并且使用了 Ajax 异步技术提高了页面响应速度；逻辑层使用 python 语言以及 Flask 框架实现服务器端的业务逻辑；数据层使用与 Flask 适配良好的 SqlAlchemy 数据库工具，提供了访问和操作数据库的接口。本文中将详细介绍系统的技术综述、需求、设计以及实现。

关键词：物资管理、Flask 框架、Bootstrap 框架

Abstract

Having a large amount of inventories to track, record, and manage, most of the current enterprises may use the traditional way which is using Excel to add, delete, and modify the records to manage inventories. Obviously, it has many disadvantages and makes the work complicated and easy to cause errors. So when the inventory takes quite a large proportion of the total physical property, inventory management becomes much more important for those enterprises, there is a need to use an inventory management system to better organize and manage the inventories.

The inventory management system design and implemented in the paper is an online management system that is applied to the company for internal use, it is the information system which is to refine the traditional way of using Excel to manage inventories, and it is the information management system that can deal with the operations of requesting, returning, scrapping, transferring, editing and searching inventories, that supports Excel by importing and exporing Excel files, that will inform related users by online messages and mails when the state of inventory changes.

The system uses three-layered structure of which the presentation layer uses HTML, CSS, JavaScript, JQuery, Bootstrap framework and Jinja2 template engine to support and the Ajax technique to improve the response speed, the logic layer uses python script language and Flask framework to implement the logical business process of the server side, and the data layer uses the SqlAlchemy tool which is the best suite for Flask to access and operate database. In the paper, the related technology, requirement, design and implementation of the inventory management system will be introduced in detail.

Keywords：Inventory Management, Flask Framework, Bootstrap Framework

目 录

图目录

表目录

第一章 引言

1.1 项目背景

当拥有一批物资需要进行跟踪、记录和管理时，大部分企业的做法通常可能是采用最传统的管理方式，即通过 Excel 增删改记录来管理，这些 Excel 表格被不断的修改、传递、加密或者是归档等等，所有的管理活动仅仅凭借着 Excel 来记录。然而，正是这样的方式存在着许多弊端，比如物资状态的更新信息不能及时反馈给公司的其他同事，操作数据容易出现信息不完整、不合理、对错号、易丢失等情况，当这些数据变得更庞杂时，更是让这项工作变得繁琐、易出错。如果需要追踪和管理的物资数量并不多时，采用传统的 Excel 来管理基本可以满足要求，但是当规模上升，当物料资源在公司的有形财产中占了比较大的份额，或者是对数据操作的安全性要求提高，对操作权限有要求等等，对于这些企业来说，物资管理就变得更加重要，这时就需要一个内部管理平台来对这些物资设备进行良好的管理。

通过传统的方式来管理物资的上海易安信研发公司 UFE 部门遇到了类似的问题。部门现有的管理物资很简单，采用 Excel 表格记录信息，通过在管理员处填写登记表格，来管理物资设备，通常数据信息都是由管理员来保管；这种管理机制虽然可以保障最基本的管理需求，但是在实际管理工作中发现了许多弊端。一是数据容易丢失，比如有一次电脑数据坏掉了，原本存有很多物资信息的表格丢失了，结果难以恢复和追溯物资的状态。还有信息不安全，在填写过程中容易对错号，把设备弄混了，或者是填进去无用的信息，填错信息等等，或者本不该有的操作被误认为可以操作，结果很混乱。还有一点不方便的就是只有管理员可以看到数据，其他用户需要通过管理员获取最新的数据信息。因此，管理员非常希望能改善现有方式存有的弊端，同时希望有检索信息的功能，能够在一张表中显示到查询结果，并且能够根据物资设备当前的状态进行安全的数据操作，其他负责维护物资的工程师则希望能够有好的消息反馈机制，能够及时通知到相关人员哪些物资状态发生了改变，而且如果采用了新的系统来管理，那么要能够兼容现有的 Excel 表格，方便从 Excel 中导入数据，或者从系统导出 Excel 表格来查

看物资信息。除此之外，需要能够批量操作以便更高效地管理物资。

在本文的工作之前，部门曾临时改造开源的管理系统作为物资管理系统，但由于其后期维护成本高，而且在界面设计上没有以用户为中心进行设计，导致系统不好用、不易维护而且不易扩展，所以只好继续用着 Excel 管理的方式。

1.2 国内外物资管理系统的发展概况

一个能够良好运作的管理系统应该具备几个要素：输入、输出、转化和控制。从而达到系统要实现的目标。对于公司的管理系统来说，目标通常就是企业想实现的目标，控制机制是企业内部的一些管理条例和办法，输入的数据以及输出的结果就是企业想通过管理系统所管理的资源。[McLeod, 2001]

物资管理系统是专职负责管理公司物料资源（包括生产品、原材料、零配件和商品等）的系统，这些物料或者向外销售，或者在公司内部使用，在这个过程中，物资管理系统会对物料资源做跟踪和记录，直到物料资源被报废后从系统中删除。通常，这个系统可以独立运行，也可以集成到企业 ERP 中。管理者通过物资管理系统可以获得物资流动、使用和消费情况，从中得到决策支持，可以更好地平衡物料资源的投入与产出。如今的物资管理系统发展都比较成熟，功能上大多都包含跟踪物资状态、进行权限操作、查看历史记录和统计报告，还有额外的功能如入库时的电子扫描、库存分析等来吸引管理者。

通常，当物资数据较少、操作简单时，公司会对其进行简单的人工管理，通过手动填写记录表格，计算，处理数据，存档等方式来管理；而稍微上了规模的物资管理则通过物资管理系统来完成。物资管理系统一般有这些形式：一、单机模式。通过单机上的文件或数据库进行管理，通过计算机电子表格，处理计算工具，或者成熟的数据库和其操作软件对企业物料资源进行记录和管理；二、C/S 模式。通过内部局域网连接企业的计算机，通过安装客户端和服务端在内部的计算机上来进行信息共享和同步，实现对物料资源的有效管理；三、基于 Web 的 B/S 模式。能够通过浏览器登录访问服务器资源，从而实现不受安装客户端的限制，不受时间和地域限制的操作，能够享受到网络带来的时效性资源和快速便捷管理。是一种对网络、信息技术和先进管理思维模式进行结合而产生的系统。[刘

肖，2001] [高凡等，2015]

如今的物资管理系统发展趋于网络化，这是因为一方面出于物资管理系统本身发展的需要，在客观上，物资管理系统要求信息实现有机集成；另一方面是对快速信息反馈的需要，通过接入网络，能够在网络可达的范围内实现快速的信息共享和同步，对于跨地区的企业合作或者是跨区域的连锁公司能够实现内部物资有效的管理都是比较明显的。通过浏览器访问，减少了安装客户端的依赖。通过网络访问到系统，实时地获取由系统通过网络推送的消息，增加了消息传播的有效性和及时性，所以依托网络化的物资管理系统，企业能够方便地进行跨地区信息共享和同步，减少交流成本，增加合作。[马智亮，2006]

1.3 本文主要工作

本文所设计与实现的物资管理系统，是一个应用于公司内部的网上管理系统；是着力于改善传统的 Excel 表格记录和管理的方式，实现物资管理自动化的信息系统；是能够对物资设备进行资源请求、归还、报废、转交、编辑、检索等操作，能够导入导出 Excel 表格，进行在线消息通知和邮件通知，让物资的状态变化及时通知到相关人员的管理信息系统。

本文工作中总结了之前部门开发的旧管理系统的问题，在设计系统以及在项目的一些决策中，比如在技术选型方面，用户体验方面都着重考虑了这两点，即系统是否拥有比较好的用户体验，以及是否容易拓展和维护。

本文的物资管理系统采用了三层结构，表现层使用了 HTML、CSS、JavaScript、JQuery、Bootstrap 框架技术以及 jinja2 模板引擎，并且使用了 Ajax 异步技术提高了页面响应速度；逻辑层使用 python 语言以及 Flask 框架实现服务器端的业务逻辑；数据层使用与 Flask 适配良好的 sqlalchemy 数据库工具，提供了访问和操作数据库的接口。在线消息机制使用了 SSE protocol，即服务器端推送消息的机制 (Server-Sent-Event) ，响应良好。[Cook, 2014] 另外，邮件通知采用了 smtplib 库，导入导出 Excel 采用了 xlwt 和 xlrd，其中 smtplib 是 python 标准库，不需要额外的安装，xlwt 和 xlrd 需要安装。

1.4 本文的组织结构

本文的组织结构如下：

第一章 引言。介绍了项目背景，国内外物资管理系统的发展概况，本文主要工作以及组织结构。

第二章 相关技术综述。介绍了物资管理系统所应用到的相关技术，包括技术的概述、作用以及在项目中如何对这些技术加以应用

第三章 物资管理系统的需求分析。给出了系统范围，分析描述了系统的功能性需求，对系统的非功能性需求也给出了相关说明。

第四章 物资管理系统的设计。重点论述了系统的设计方案，从高层架构到低层设计，给出了各个设计阶段的设计说明。

第五章 物资管理系统的实现。在系统分析与设计的基础上，重点阐释了系统的关键实现，给出了关键的实现代码。

第六章 总结与展望。总结工作，并且就今后的发展作了进一步展望。

第二章 相关技术综述

2.1Flask 框架

物资管理系统采用了 Flask 框架技术做服务器端的后台开发，Flask 框架是一款轻量级的支持网络开发的 python 框架，类似的还有 Django, Tornado 等框架，之所以选择用 Flask 框架，考虑的因素有：一、部门团队的同事们普遍熟悉 Python, 而且有 Flask 框架开发的经验，便于理解和后期维护；二、Flask 框架被业界认可，其特点在于灵活、轻量级，基于 Werkzeug、jinja 2 和一些知名的开源库。拥有内置的服务器和调试器，集成了一些单元测试，拥有 RESTful 请求适配，支持安全的 cookies 访问（客户端 session），100%的 WSGI 1.0 兼容，文档齐全，学习起来容易、灵活。[叶锋, 2015] 三、Flask 的 jinja2 模板引擎用起来灵活、方便，利于组织前端页面，提高前端代码的重用率和简洁性。这样可以帮助理清代码，而良好的代码结构有利于开发和维护。

2.1.1 Flask 框架简介

Flask 框架因其实现的核心简单且具有良好的扩展性而被称作是微型框架（microframework），由 python 写成，基于 Werkzeug toolkit 库和 Jinja2 模板引擎来实现，werkzeug 和 jinja2 均由 python 实现，且都是开源的（BSD licensed），werkzeug 是比较强大的 WSGI（Web Server Gateway Interface） 工具箱，jinja2 是使用起来灵活的模板引擎，支持传参，模板继承，以及一些循环、判断语句。因此，Flask 也是开源的，在上述两个强大的开源工具支持下，能够帮助程序员快速实现一个轻型的 web 应用。[Flask, 2016]

Flask 之所以为微型，是因为 Flask 不会替开发者做决策，例如使用什么数据库。Flask 对其他的库的依赖很少，即便是 jinja2，在开发者需要的时候，也是可以更换为其他的模板。[Grinberg, 2015]

Flask 安装简单，在 pip 的帮助下，只需要执行 pip install Flask 即可。Flask

拥有丰富的插件，插件的安装也大多简单，比如安装 sqlalchemy 数据库工具，只需要执行 pip install Flask-sqlalchemy 即可。

Flask 所约定俗成的部分是，在应用的根目录下，需要创建两个文件夹，一个需要命名为 templates，存放 html 模板文件，还有一个需要命名为 static，存放 css, js, 图片和其他静态资源。当然这约定俗成的部分在开发者需要的时候也可以改变。[Flask, 2016]

2.1.2 Flask 框架应用

最简单的 Flask 可运行版本是一个单文件，如下图所示，直接运行该文件，即可在浏览器中输入地址 http://localhost:5000/看到相关输出。

```
from flask import Flask
app_instance = Flask(__name__)

@app.route("/")
def hello_world():
    return "Hi World, Hello Wendy!"

if __name__ == "__main__":
    app_instance.run()
```

图 2.1 Flask 框架实现的最简运行版本

Flask 的一个设计是：用一个可以调用的显式的对象来启动服务，在 Flask 中是创建了一个 Flask 类的实例，而一个 Flask 类的实例创建时，必须要将当前模块的名字传入 Flask，在上述的例子以及物资管理系统中，我们所用的是 Flask(__main__)，所以当__main__函数运行时，实例被创建，服务器被启动，在浏览器中访问对应的 url 地址会有响应。[Flask, 2016]

而在 Flask 框架的内部，主要的工作是由 Werkzeug 的路由系统来匹配 url 请求和服务器的响应，用 jinja2 模板引擎来渲染页面。查看了 Flask 源码，可以得出如图 2.2 所示的结构图。

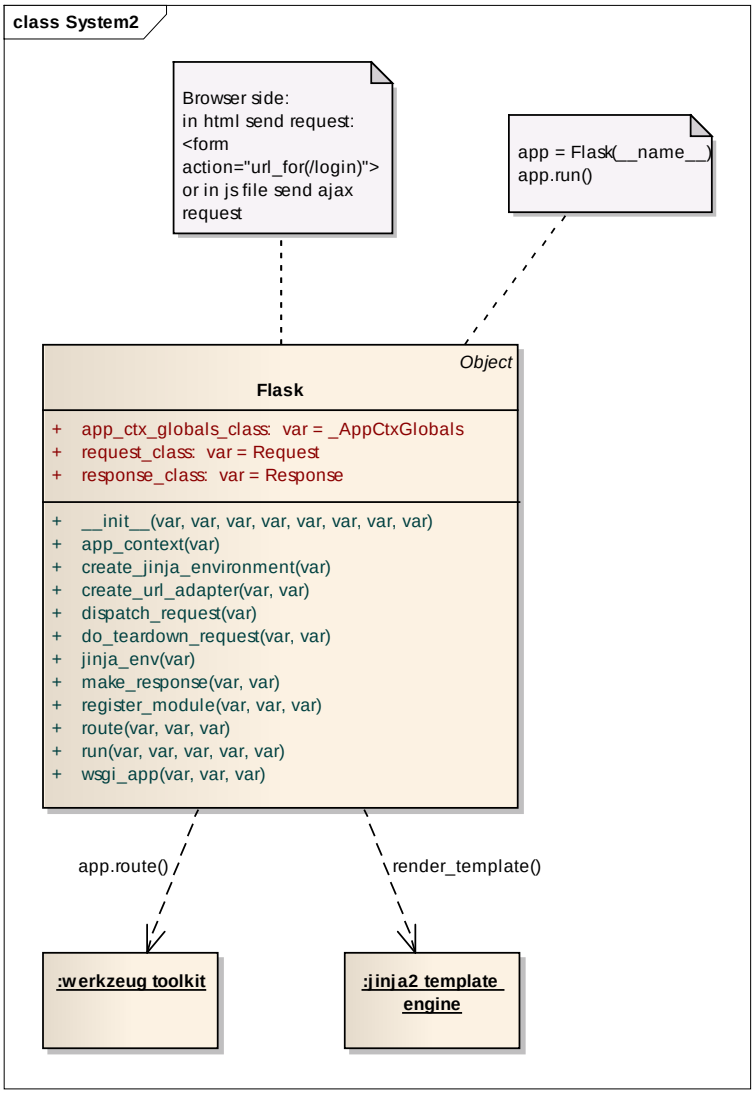

图 2.2 Flask 运作原理

当利用 Flask 框架构建一个较大型的 web 应用时，应该将系统的整体结构调整为如图 2.3 所示的工程目录结构，将启动服务器放在最外层的 runserver.py 文件中，其他系统的模块实现都置于和 runserver.py 平行的目录文件夹下。

```
/yourapplication
    /runserver.py
    /yourapplication
        /__init__.py
        /views.py
        /static
            /style.css
        /templates
            /layout.html
```

图 2.3 Flask 应用的工程目录结构

这种结构的好处除了将系统组织成模块，利于系统的拓展外；还有一点就是在 init 文件中创建 Flask 实例，方便 runserver.py 文件和其他模块的文件从中导入该实例，从而调用 Flask 提供的接口。

2.2 Bootstrap 框架

2.2.1 Bootstrap 简介

Bootstrap 是 HTML、CSS、JavaScript 的前端开发框架，也是开源的框架。Bootstrap 以 JQuery 插件的形式提供了很多 JavaScript 组件，也扩展了现有组件的功能。除了 HTML 常规的组件，还提供了额外的组件方便使用，比如设计风格统一的 glyphicon 的小图标，为界面设计带来美观、协调和一致的风格。[Spurlock, 2013] 除此之外，Bootstrap 让实现一个响应式网站更加方便。一句话概括响应式网页设计，就是针对任意设备对网页内容进行好的布局的一种显示机制。[Frain, 2013]

2.2.2 Bootstrap 应用

Bootstrap 应用起来简单，大多数的组件功能和风格均已被框架定义好，所以使用的时候，直接将元素的 class 属性赋值为 Bootstrap 定义的类即可，比如：<table class="table table-striped"></table>，像这样用 Bootstrap 装饰后的表格将显示 Bootstrap 定义的默认表格的外观样式以及叠加了深条纹间隔的样式。

另外，在 html 的 head 标签中需要加入 meta 标签，使得 Bootstrap 能够得到正确的渲染。一般来说，定义显示宽度为设备的宽度，初始缩放比例为 100%：<meta name="viewport" content="width=device-width, initial-scale=1">。至于 Bootstrap 的安装则比较简单，下载好 Bootstrap 包，拷贝需要的文件放在项目中，再在 html 中加入对 Bootstrap 的引用即可。

2.3 前端与后台的交互

基于 Flask 框架和 Bootstrap 框架分别构建了后台和前端之后，前后台的交

互，也就是一个请求和响应的序列就成为关注点。以切换页面的操作为例：当用户点击导航中的 Inventory 按钮，系统作出响应，返回 Inventory 页面呈现在浏览器端，在系统内部，这个响应的流程如图 2.4 所示：

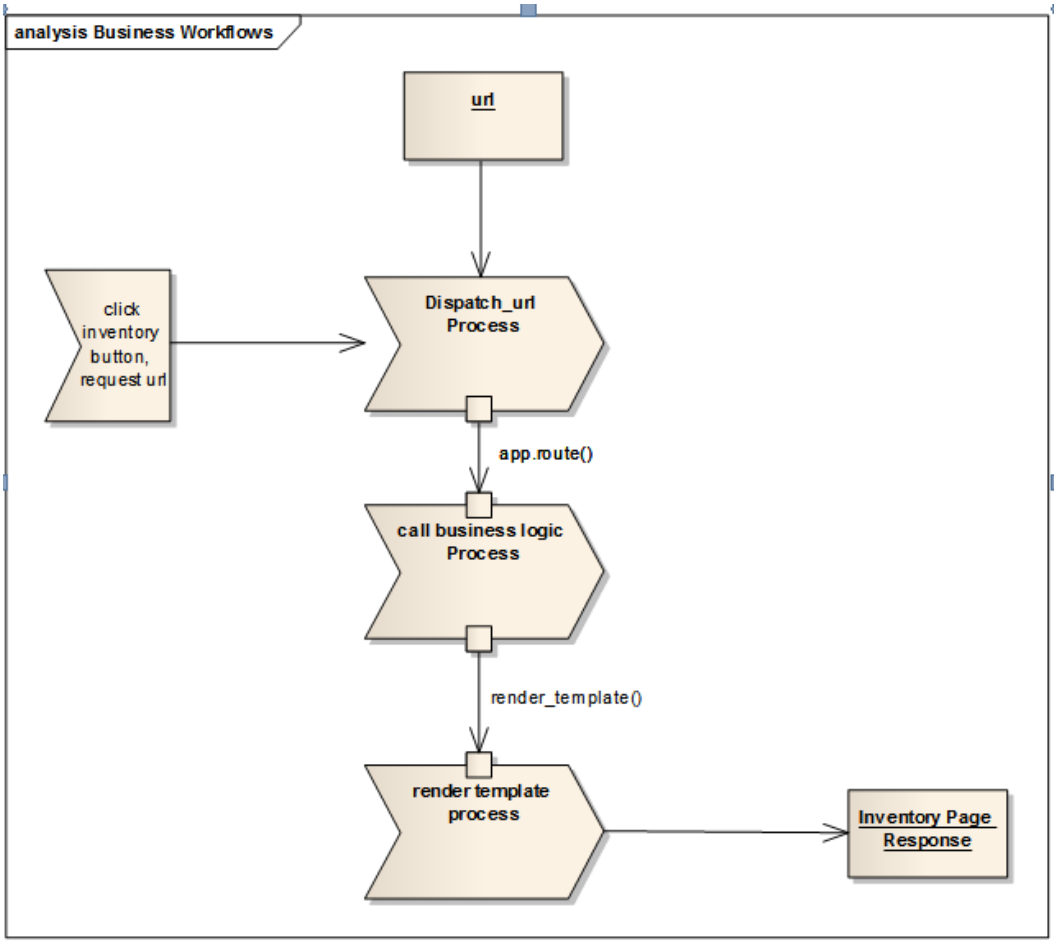

图 2.4 系统内部的响应流程

在前端，我们可以通过 form 表单来向服务器发送数据、请求资源，或者借用 Javascript 来实现一个按钮触发事件，在事件处理中通过 Ajax 请求来向服务器发出异步请求（也可以设置为同步请求，即等待响应结果）。也就是利用 Ajax 在用户和服务器之间建立一个中介。[Negrino, 2012]

在后台，通过 Flask 框架的装饰方法@app.route()来路由请求地址，被该装饰符所装饰的方法将被调用，执行相关业务逻辑，最后返回一个 Response 结果，通常一个 Response 可以是单纯的文本文字，一个 render_template()调用后返回的页面，一个重定向的地址，一个 json 数据，或者跟据其他需要而构建的 Response 对象，例如在物资管理系统中，为了实现服务器推送消息的机制，返

回了一个被监听的”text/event-stream”类型的 Response 对象。

2.4 SQLAlchemy 数据库工具

SQLAlchemy 是一款开源的软件，提供 SQL 工具包和对象关系映射的服务。SQLAlchemy 是用 python 实现的，实现了类似于 Hibernate 框架的映射机制。

SQLAlchemy 把数据库看作是关系型的，不仅是表格的集合，数据记录不仅可以从表中选出，也可以从关系中以及嵌套的 select 语句中选出，因此 SQLAlchemy 提供了这些接口服务。SQLAlchemy 实现了对象关系映射（ORM）的服务，提供了数据映射模式，能够方便地把类映射到数据库。[Noab, 2009]

Flask-SQLAlchemy 给 Flask 框架提供了 SQLAlchemy 的支持，这样在 Flask 框架中利用 SQLAlchemy 操作数据库就变得比较简单，首先，在类中声明数据模型，在查询记录前先向数据库添加数据：创建数据模型类的 python 对象，添加到 session 中，最后提交 session。这里的 session 不是表现层的，而是 Flask-SQLAlchemy 中的 session。Flask-SQLAlchemy 为数据类插入了 query 的成员变量，代表了所有记录集合的对象，而这个 query 对象进一步提供了多种方法来方便查询：获取所有记录、获取首条记录、过滤和通过某个字段过滤等等。

映射对象是通过 Flask-SQLAlchemy 创建的 SQLAlchemy 实例来初始化每一个需要映射到数据库的数据类，所有的数据类都有一个父类：db.Model。在类中可以声明要映射到的数据库表的名称，也可以用默认的将类名转换成小写的数据库表的名称。SQLAlchemy 还提供了数据库表之间的一对一、一对多、多对多的关系，这些都可以利用 Flask-SQLAlchemy 提供的接口来实现。

2.5 Jinja2 模板引擎

Jinja2 是仿照 Django 模板的 python 模板引擎。拥有强大的 HTML 转义系统，支持模板继承，能够及时地编译 python 代码，可配置，易调试。Jinja2 是通用的模板引擎，不只是能够输出 HTML 或者 XML 文件，还能渲染出邮件、CSS、JavaScript 文件等等。

模板包含了变量和表达式，这两者在经过引擎渲染之后会被替换成相应计算的值，用分隔符：{%...%}和{{…}}来表达，前一种用来执行相关语句，后一直用来计算表达式。变量可以从应用传递到模板，如果变量是一个对象，进一步访问该变量的属性值也是被允许的。此外，Jinja2 还有内置过滤器，多个过滤器可以一起形成链式的过滤效果；内置测试器可以帮助测试一些简单的数据；关于转义，也提供了适用于单条语句和整块语句的做法。模板还提供控制结构如 if 语句，for 语句和宏语句等，置入第一种分隔符内，提供程序流的控制。[Jinja2, 2008]

模板继承的功能比较强大，可以帮助组织前端页面代码的结构，重用一些共享的代码，构建出应用模板的“骨架”，通过定义子模板覆盖父模板的块来拓展具体的页面细节。一般，都会定义一个基本模板 base.html，将一些头元素、链接的文件放在这里面，而子模板实现时覆盖该基本模板预留的块，再经过引擎渲染来构建出完整的页面。

2.6 本章小结

本章对本文工作中所使用到的技术做了综述。给出了选择 Flask 框架的原因，Flask 框架的介绍和应用，Bootstrap 框架的介绍和应用，本文方案下前端与后台交互机制的介绍，SqlAlchemy 数据库工具的介绍以及 Jinja2 模板引擎的介绍。

第三章 物资管理系统的需求分析

3.1 系统范围

本文的物资管理系统主要解决用 Excel 管理物资的弊端：现有的物资管理采用 Excel 表格记录信息，通过在管理员处填写登记表格，来管理物资设备，数据信息都是由管理员来保管；虽然可以保障最基本的管理需求，但是容易丢失数据，而且在填写过程中容易对错号，或者是填进去无用的错误的信息等等，或者对物资进行了不合理的操作。而且只有其他用户需要通过管理员来获取最新的数据信息。因此，本文的物资管理系统改善传统的管理方式，能够导入导出 Excel，能够检索信息，能够批量对物资进行操作，能够根据物资当前的状态对其操作，能够在物资状态变化时及时通知到相关的用户。

3.2 需求分析

该系统的涉众有：小组领导，系统管理员，系统用户。对于小组领导来说，更关心项目的时间、进度和花费的人力物力财力，希望能够以较小的成本完成基本满足工程师需求的物资管理系统；对于系统管理员来说，看重的是每一个物资的信息是否完整、正确，关心每一项物资的状态及历史信息，以及状态更新是否能及时通知，是否方便导入导出 Excel 表格来兼容传统管理方式；对于系统的普通用户来说，则关心的是否方便查询设备，方便管理自己负责的或者借用的物资。

结合以上的涉众分析，在和用户协商后，对这些需求做了优先级排列，包括：物资记录的增删改查，对物资的请求，归还，报废，转让处理，导入导出 Excel 表格，用户记录的增删，消息通知，邮件通知以及历史记录查看。

通过绘制用例图，描述了物资管理系统应该满足的用户需求，如图 3.1 所示是物资增删改查的用例图，在编辑物资的用例中，又包含了两个子用例，即消息通知和邮件通知。

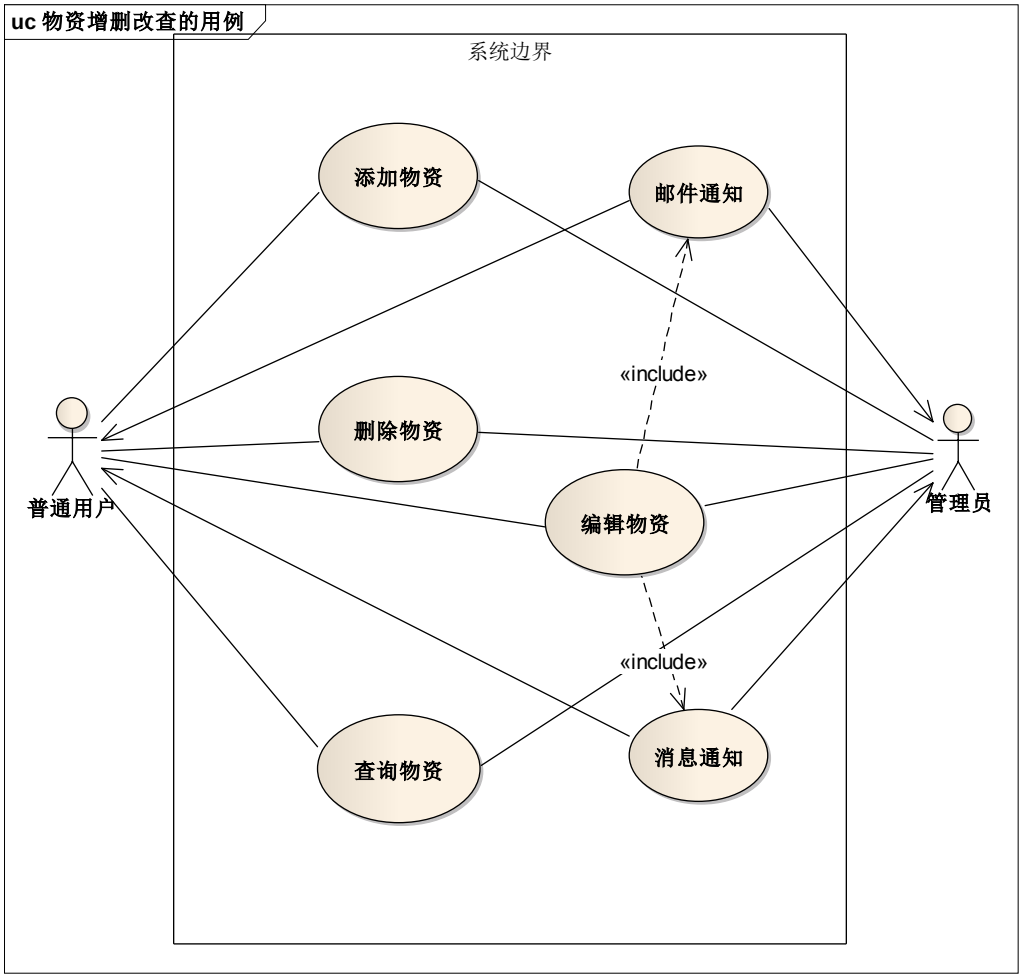

图 3.1 物资增删改查的用例图

添加物资的功能可以被普通用户和管理员使用，系统要引导用户正确的填写信息完成添加的操作，在添加错误时给出恰当提示，添加物资成功后，系统应该能够在界面上显示出新添加的物资，具体的用例描述请见表 3.1。

表 3.1 添加物资的用例描述

用例：添加物资
范围：物资管理系统
级别：用户目标
主执行者：普通用户或管理员
相关人员和利益：普通用户和管理员，能够正确适当地添加物资
前置条件：用户身份验证成功，页面加载完成，当前界面处于物资管理页面
最小保证：返回到物资管理页面

成功保证：添加物资完毕，返回到物资管理页面，添加成功的物资出现在列表中

触发事件：用户或管理员选择“添加物资”

主成功场景：

1. 普通用户或管理员选择“添加物资”；
2. 系统提供添加物资的界面；
3. 用户输入物资具体信息，选择确认添加；
4. 系统验证了输入信息的类型和格式正确，并向数据库中添加该项物资；
5. 添加成功，系统返回成功添加的消息
6. 系统返回物资管理页面，并将新添加的物资展示在页面中。

扩展：

3.a 用户取消添加物资；

3.a.1 系统返回物资管理页面，终止用例；

4.a 系统验证输入信息有误，提示用户修改；

5.a 添加失败，系统返回失败的消息；

5.a.1 系统返回物资管理页面，终止用例。

技术和数据变化列表：

3.a 用户使用 RFID 识别器扫描物资；

3.a.1 系统向数据库添加该项物资；

删除物资的功能可以被普通用户或者管理员使用，系统允许用户批量地删除物资，因为是从数据库删除掉物资，所以系统要能够在删除之间询问用户的意愿，确认用户不是误删。删除物资的用例描述如表 3.2。

表 3.2 删除物资的用例描述

用例：删除物资

范围：物资管理系统

级别：用户目标

主执行者：普通用户或管理员

相关人员和利益：普通用户和管理员，能够有效地删除某些物资，能够批量删除

前置条件：用户身份验证成功，页面加载完成，当前界面处于物资管理页面

最小保证：返回到物资管理页面

成功保证：删除完毕，返回到物资管理页面，删掉的物资信息也从界面消除

触发事件：用户或管理员选择“删除物资”

主成功场景：

1. 普通用户或管理员选中一项物资，选择“删除物资”；
2. 系统提供确认删除的界面；
3. 用户选择确认删除；
4. 系统从数据库中删除该项物资；
5. 删除成功，系统返回成功删除的消息
6. 系统返回物资管理页面，该项物资从展示页面中消除。

扩展：

1.a 用户选中多项物资，选择“删除物资”；

3.a 用户选择取消；

3.a.1 系统返回物资管理页面，终止用例；

5.a 删除失败，系统返回失败的消息；

5.a.1 系统返回物资管理页面，终止用例。

技术和数据变化列表：

无

编辑物资信息的功能可以由普通用户或者管理员来完成，同样系统要引导用户填写正确的信息，在更新信息时检查数据类型，对不可随意更改的信息做保护，等等。编辑成功后，需要对相关的用户发出消息和邮件通知，相关的用户指的是物资的负责人和使用者（如果物资处于使用状态）。这样，物资的信息更改可以及时通知到相关用户，不仅能够提高管理的效率，而且如果出现错误，能够及时被相关用户发现。编辑物资的用例描述如表 3.3。

表 3.3 编辑物资的用例描述

用例：编辑物资

范围：物资管理系统

级别：用户目标

主执行者：普通用户或管理员

相关人员和利益：普通用户和管理员，能够正确适当地修改某项物资的信息

前置条件：用户身份验证成功，页面加载完成，当前界面处于物资管理页面

最小保证：返回到物资管理页面

成功保证：编辑完毕，返回到物资管理页面，修改后的物资显示在界面中

触发事件：用户或管理员选择“编辑物资”

主成功场景：

1. 普通用户或管理员选中一项物资，选择“编辑物资”；
2. 系统提供编辑物资的界面；
3. 用户输入修改信息，选择确认编辑；
4. 系统验证了输入信息的类型和格式正确，并在数据库中更新了该项物资信息；
5. 编辑成功，系统返回成功编辑的消息
6. 系统返回物资管理页面，编辑后的物资信息展示在页面中。
7. 系统向相关用户发送消息通知和邮件通知

扩展：

3.a 用户取消编辑；

3.a.1 系统返回物资管理页面，终止用例；

4.a 系统验证输入信息有误，提示用户修改；

5.a 编辑失败，系统返回失败的消息；

5.a.1 系统返回物资管理页面，终止用例。

技术和数据变化列表：

无

查询物资的功能可以被普通用户和管理员使用，而且查询物资以便捷为前提，所以系统要能够把查询物资的功能置于醒目的地方，对于用户来说，输入关键字查询后，希望看到和出现这个关键字的所有条目，所以系统要对物资的每一

个字段都进行关键字的查找，才能满足用户的需要。具体用例描述见表 3.4。

表 3.4 查询物资的用例描述

用例：查询物资
范围：物资管理系统
级别：用户目标
主执行者：普通用户或管理员
相关人员和利益：普通用户和管理员，能够快速地根据关键字查询到物资
前置条件：用户身份验证成功，当前界面有查询物资的面板
最小保证：返回到触发事件前的页面
成功保证：查询完毕，展示搜索列表
触发事件：用户或管理员选择“查询物资”
主成功场景：
1. 普通用户或管理员输入关键字，选择“查询物资”;
2. 系统搜索关键字，返回查询结果;
3. 系统跳转到搜索结果页面，查询结果展示在页面中。
扩展：
1.a 用户未输入关键字，直接选择“查询物资”;
1.a.1 系统停留在当前页面，终止用例;
技术和数据变化列表：
无

类似于物资的增删改查，关于用户管理，由于最终系统应用于公司要接入公司的账户管理系统，所以这一块没有对用户进行编辑和查找的需求，只是做增删即可，而且主要用在系统测试时添加和删除测试数据，最终系统接入公司账号管理系统后，需要修改添加的方式，即从账户管理系统中搜索并添加这个用户。鉴于项目初期系统的完整性，故先实现简单的用户增删需求。如图 3.2 所示是用户增删的用例图。

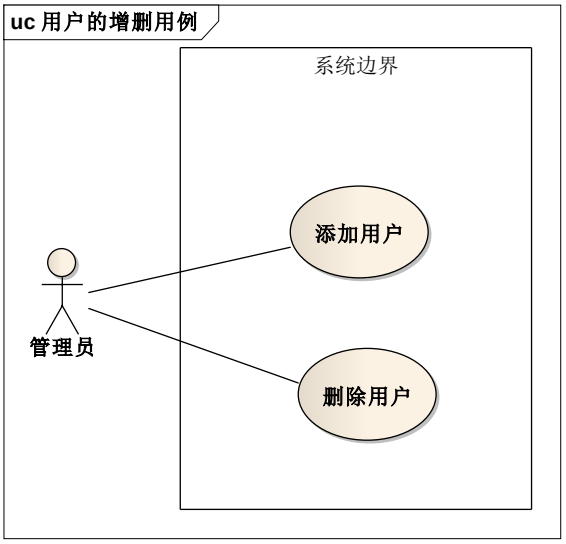

图 3.2 用户增删的用例图

在物资管理系统中，对物资的操作处理有请求、归还、转让和报废，并且包含了两个子用例，即消息通知和邮件通知。此外，跟据物资当前的状态不同，能够进行的操作也不同：

表 3.5 物资状态及操作对应表

	可用（Available）	使用中（Used）	报废（Scraped）
资源请求	√	×	×
资源归还	×	√	×
转让	√	√	×
报废	√	×	×

另一方面，物资的状态切换是通过这些操作来进行的，即：可用状态通过成功的资源请求切换到使用中状态；可用状态通过报废切换到报废状态；可用状态通过成功资源归还切换到可用状态；当物资成为报废状态时，这些操作都禁止。当状态为可用或者使用中的时候，可以将物资转到某一用户名下,该用户即成为这些物质的负责人。

另外，报废物资的用例只能由管理员来操作，普通用户没有权限对物资进行报废的操作。关于物资操作的用例图如图 3.3 所示：

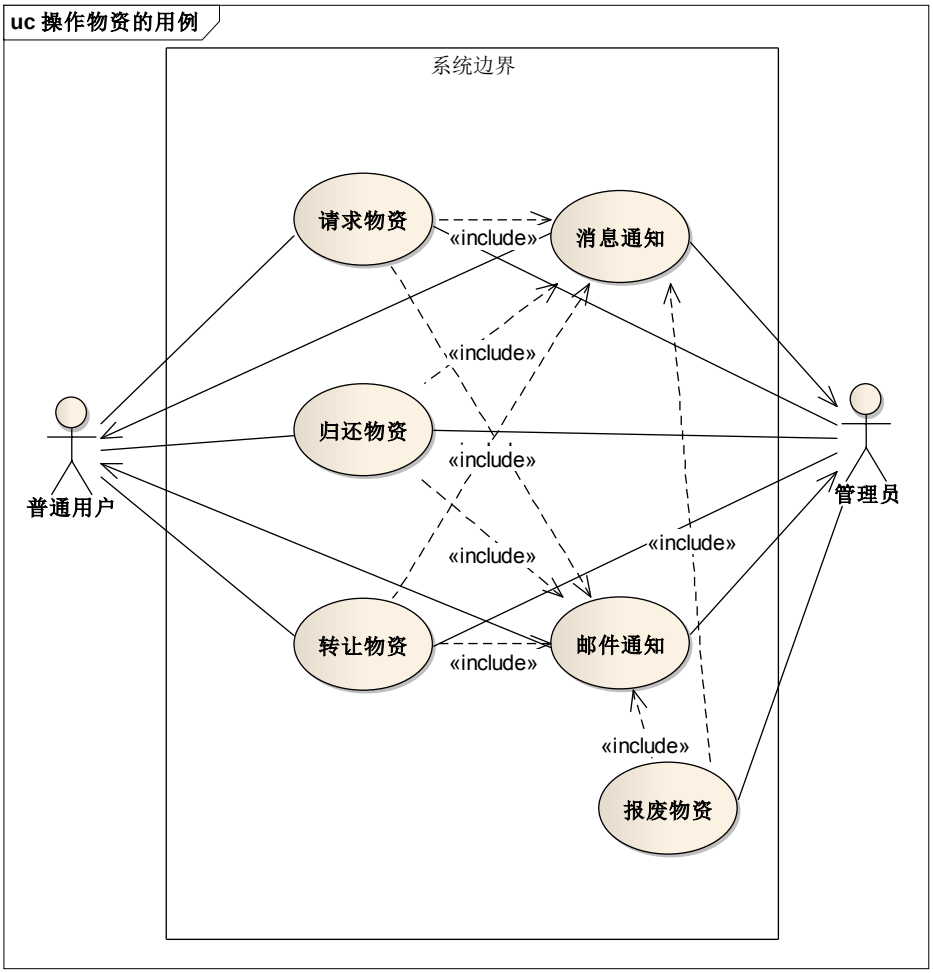

图 3.3 物资的请求、归还、转让和报废的用例图

请求物资和归还物资属于类似的用例，普通用户和管理员都可通过物资管理系统进行物资的请求和归还，具体的用例描述如下：

表 3.6 请求和归还物资的用例描述

用例：请求物资、归还物资
范围：物资管理系统
级别：用户目标
主执行者：普通用户或管理员
相关人员和利益：普通用户和管理员，能够有效地请求（或归还）物资
前置条件：用户身份验证成功，页面加载完成，当前界面处于物资管理页面
最小保证：返回到物资管理页面
成功保证：请求完毕，返回到物资管理页面，物资状态变化

触发事件：用户或管理员选择“请求物资”（或“归还物资”）

主成功场景：

1. 普通用户或管理员选中一项或多项物资，选择“请求物资”（或“归还物资”）；
2. 系统验证了用户身份和物资状态符合要求，向数据库更新物资状态信息，添加历史记录
3. 请求（或归还）物资成功，系统返回成功的消息
4. 系统返回物资管理页面，物资状态改变，使用者信息更新
5. 系统通过在线消息和邮件通知相关用户

扩展：

2.a 系统发现物资状态不适合请求（或归还）或者用户身份不能够请求（或归还）当前物资，不提供“请求物资”（或“归还物资”）的接口，终止用例；

3.a 请求失败，系统返回失败的消息；

3.a.1 系统返回物资管理页面，终止用例。

技术和数据变化列表：

无

报废物资只能被管理员使用，在管理员确认一项或一批物资报废后，可以通过物资管理系统对这些物资进行报废操作，关于报废物资的用例描述见表 3.12。物资变成报废状态后，用户就不能再对其进行请求和归还操作，但是系统还继续维护报废物资，直到物资被删除。

表 3.7 报废物资的用例描述

用例：报废物资

范围：物资管理系统

级别：用户目标

主执行者：管理员

相关人员和利益：管理员，能够有效地报废物资

前置条件：管理员身份验证成功，页面加载完成，当前界面处于物资管理页面

最小保证：返回到物资管理页面

成功保证：报废操作完毕，返回到物资管理页面，物资状态变为报废

触发事件：管理员选择“报废物资”

主成功场景：

1. 管理员选中一项或多项物资，选择“报废物资”；
2. 系统验证了管理员身份和物资状态符合要求，向数据库更新物资状态信息，添加历史记录
3. 报废物资成功，系统返回成功的消息；
4. 系统返回物资管理页面，物资状态改变
5. 系统通过在线消息和邮件通知相关用户

扩展：

2.a 系统发现物资状态不适合报废或者当前用户身份不能够报废当前物资，不提供“报废物资”的接口，终止用例；

3.a 报废操作失败，系统返回失败的消息；

3.a.1 系统返回物资管理页面，终止用例。

技术和数据变化列表：

无

转让物资的功能确切地来说能够被该物资的负责人和管理员所使用，所以系统需要检查当前用户是否是该物资的负责人或者是管理员，而且对接收转让的用户身份也要做验证，保证转让是有效的。转让物资的用例描述如下：

表 3.8 转让物资的用例描述

用例：转让物资

范围：物资管理系统

级别：用户目标

主执行者：普通用户或管理员

相关人员和利益：普通用户和管理员，能够正确适当地转让物资给其他用户

前置条件：用户身份验证成功，页面加载完成，当前界面处于物资管理页面

最小保证：返回到物资管理页面

成功保证：转让完毕，返回到物资管理页面，物资的负责人信息更新

触发事件：用户或管理员选择“转让物资”

主成功场景：

1. 普通用户或管理员选中一项或多项物资，选择“转让物资”；
2. 系统验证了用户身份和物资状态符合要求，提供转让信息输入的界面；
3. 用户输入负责人信息，选择确认转让；
4. 系统验证了负责人信息正确，在数据库中更新物资的负责人信息，添加历史记录；
5. 转让成功，系统返回成功的消息
6. 系统返回物资管理页面，物资的负责人信息在界面上更新。

扩展：

2.a 系统发现物资状态不适合转让或者用户身份不能够转让当前物资，不提供“转让物资”的接口，终止用例；

4.a 系统验证输入信息有误，提示用户修改；

5.a 转让失败，系统返回失败的消息；

5.a.1 系统返回物资管理页面，终止用例。

技术和数据变化列表：

无

查看历史记录的功能是面向管理员的，管理员可以按照时间发生的倒序来查看到所有物资的历史记录，包括每一个操作和状态变化的历史，后期版本中可以加入更多样的查看方式，比如按照每一个物资来查看历史记录，选择时间段来查看，按照操作类型来查看记录等等。

导入导出 Excel 的功能面向所有用户，在导入 Excel 的过程中，系统要能够检查文件类型，在导入的过程结束后，给出导入成功和失败的物资说明；关于导出 Excel，则提供给用户一个下载文件的接口。导入导出 Excel 的用例图如图 3.4。

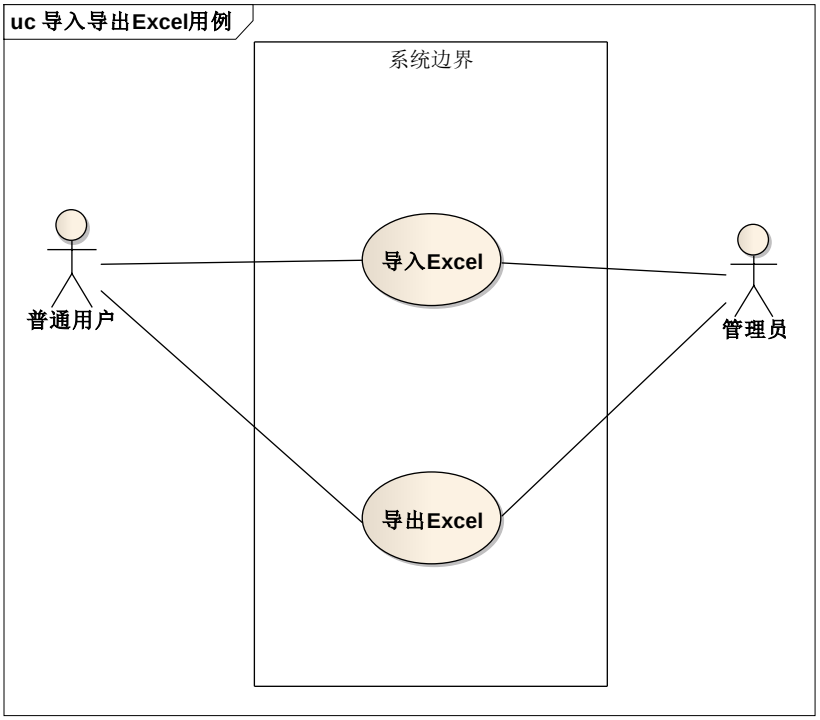

图 3.4 导入导出 Excel 的用例图

导入 Excel 的功能目的在于添加物资，所以系统要能够处理失败的情况而不中断导入过程，尽可能多的导入信息正确的物资，相关用例描述如下：

表 3.9 导入 Excel 的用例描述

用例：导入 Excel
范围：物资管理系统
级别：用户目标
主执行者：普通用户或管理员
相关人员和利益：普通用户和管理员，能够有效地从 Excel 导入物资数据
前置条件：用户身份验证成功，页面加载完成，当前界面处于物资管理页面
最小保证：返回到物资管理页面
成功保证：导入数据完毕，返回物资管理页面，导入的物资信息显示在界面
触发事件：用户或管理员选择“导入 Excel”
主成功场景：
1. 普通用户或管理员选择“导入 Excel”；
2. 系统提供导入 Excel 文件的界面；

3. 用户选择导入的 Excel 文件，选择确认导入；

4. 系统验证了文件类型正确、文件上传成功，向数据库中添加导入的物资数据；

5. 导入成功，系统返回成功的消息

6. 系统返回物资管理页面，新导入的物资信息展示在页面中。

扩展：

3.a 用户取消；

3.a.1 系统返回物资管理页面，终止用例；

4.a 系统验证输入文件类型有误或者上传文件失败，提示用户重新选择文件；

5.a 导入失败，系统返回失败的消息；

5.a.1 系统返回物资管理页面，终止用例。

技术和数据变化列表：

无

导出 Excel 的功能应该置于比较明显的地方，方便用户日常使用，因为目前用户比较熟悉 Excel，也可以利用 Excel 做一些报告，帮助领导、管理员和其他用户过渡到物资管理系统。这在项目投入应用的初期是比较重要的。具体的用例描述如表 3.10。

表 3.10 导出 Excel 的用例描述

用例：导出 Excel

范围：物资管理系统

级别：用户目标

主执行者：普通用户或管理员

相关人员和利益：普通用户和管理员，能够从系统导出物资数据到 Excel 文件

前置条件：用户身份验证成功，页面加载完成，当前界面处于物资管理页面

最小保证：返回到物资管理页面

成功保证：导出数据完毕，返回物资管理页面，导出的 Excel 文件存放在默认下载路径下

触发事件：用户或管理员选择“导出 Excel”

主成功场景：

1. 普通用户或管理员选择“导出 Excel”；
2. 系统将物资数据导出到 Excel 表格；
3. 导出成功，系统将导出的 Excel 文件以下载的形式存放到默认下载路径
4. 系统停留在物资管理页面

扩展：

3.a 导出失败；

3.a.1 系统提示导出数据失败，终止用例；

技术和数据变化列表：

无

最后，对系统的非功能需求也做了分析。在管理员和其他用户的描述中，多次提到系统要好用，能够在后期比较容易地做维护和扩展，因此，对物资管理系统来说，需要满足的非功能需求有三点：一是易用性，用户要能够方便地通过他们可用的途径来访问系统，比如通过浏览器访问，因此系统也不应该局限于特定的浏览器，而且用户界面应该要简洁美观，要有较好的用户体验，比如可以批量操作，界面上所有的按钮是所见即所得的。二是可维护性，系统应该能够方便地配置，比如把可配置的数据单独放在一个文件中，代码应该能够易被理解和维护，所以应该有较好的注释以及采用部门同事熟悉的编程语言，模块划分得当容易被维护等等。三是可扩展性，系统要能够方便扩展业务功能，比如支持统计的业务，并且在技术上的选择也要考虑支持可扩展性。

3.3 本章小结

本章应用面向对象的方法对需求进行了分析，给出了系统范围，通过用例图和用例描述对需求进行了详细的描述，并且对系统的非功能性需求给出了说明。本章中的用例在与用户确认后最终在需求方面达成一致。

第四章 物资管理系统的设计

4.1 架构设计

根据第三章的需求分析，物资管理系统可进行如图 4.1 所示的功能模块划分，物资管理系统首先分为三个子模块分别是：物资管理、用户管理和其他辅助管理。在物资管理的模块下继续划分为物资操作管理、导入导出表格、在线消息通知和邮件通知四个子模块，在用户管理的模块下继续划分出用户身份验证和用户消息管理的子模块，而其他辅助管理模块则是可扩展的业务模块，使物资管理系统成为更好的管理信息系统，其中包括了历史记录子模块、权限管理子模块和统计数据子模块等等。

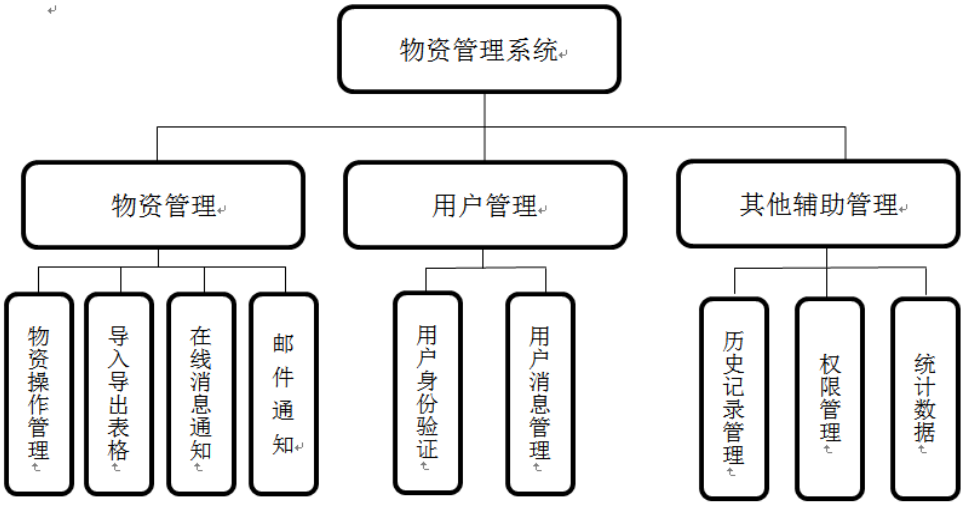

图 4.1 系统功能模块划分

根据技术选型、需求分析和功能模块划分，物资管理系统设计为采用三层结构。架构设计如图 4.2 所示。每一层只依赖下一层所提供的服务。这样分层的好处在于支持系统的增量式开发，是可改变和可移植的。[Sommerville, 2010]

表现层主要负责用户界面以及前端交互，向用户及管理员提供了一组界面，能够在界面上对物资进行操作，进行正确的操作之后，物资状态发生改变，改变将显示到当前页面，并且将会通知到相关用户。管理员能够看到所有用户，并进行增删操作。用户还能够切换到自己的设备页面，查看并操作自己拥有的或者自

已借用的这些物资设备。表现层能够向下层的业务逻辑层发起调用，通过 form 表单请求或者是 Ajax 请求的方式来实现。

逻辑层主要负责响应表现层发送的请求，进行业务逻辑处理，并返回响应结果。在物资管理系统中，业务逻辑层能够适配 url 请求，对物资的操作进行处理，一般是通过调用数据层的数据操作方法和其他单独提供的服务逻辑来实现业务，最终返回一个表现层所需要的合理数据，可能是一个页面，可能是一个 json 数据，可能是重定向地址，或者可能是其他 Response 对象。

数据层主要负责数据操作与数据持久化。数据操作是提供给业务逻辑层调用的接口，数据持久化使用了 sqlalchemy 数据库工具。

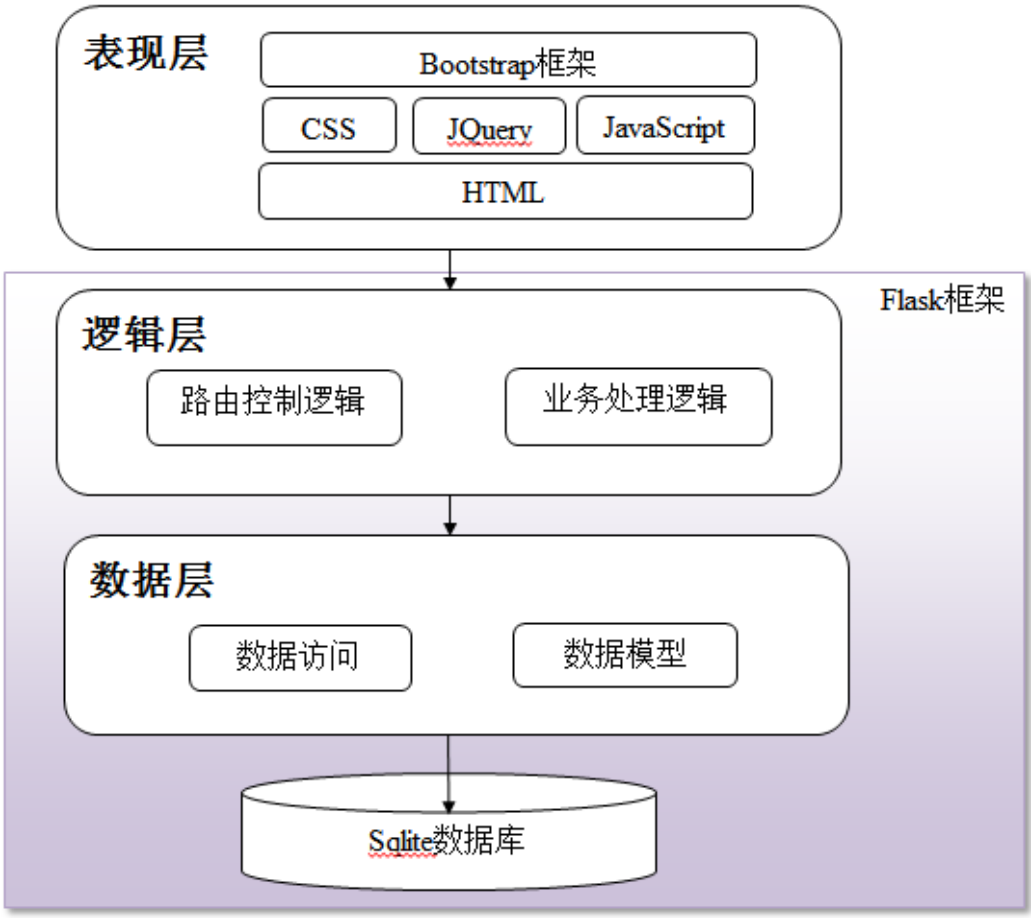

图 4.2 系统的三层结构

4.2 模块设计

在功能模块划分和架构设计基础上，进一步按照数据划分，详细设计了系统得以运行和实现的各个模块，分为：初始化模块、配置模块、启动模块、DAO模块、数据初始化模块、数据模型模块、服务模块和路由模块。其中，初始化模块、配置模块、启动模块是为 Flask 框架应用所设计；数据初始化模块则为数据库所设计。

数据模型模块、数据访问（DAO）模块对应着数据层。数据模型模块中，用各个类去描述各个数据表结构，其余的工作，创建数据库和数据库表，做表和数据类的映射等等，都利用 Flask sqlalchemy 的 model 实例去做，可以另外单独做一个数据库初始化的模块来进行。数据访问（DAO）模块用于操作数据，在系统中，通过操作对象来实现操作数据库。

剩下的服务模块和路由模块，则对应着业务逻辑层。路由模块负责主要的业务逻辑，对浏览器请求的 url 地址做路由和响应，以及一些跳转控制。服务模块负责相对比较独立的业务逻辑的提供，包括但不限于：导入导出 Excel 文件服务、SSE 服务、发送邮件的服务等等。其中，路由模块承担了大部分的服务器端的业务逻辑控制，而把较为独立的业务委托给服务模块，这些业务包括了导入导出 Excel 的后台解析工作，服务器发送消息的格式定义，邮件定制和发送的服务等。

由于 Flask 规定必须有 static 和 templates 包来引导 jinja2 渲染页面，因此表现层设计为 static 包用来组织 css、js 文件等静态文件；templates 包则用来组织 html 模板文件。

具体每个模块设计如下。如图 4.3 所示是系统的开发视图，图 4.4 所示是系统的概念类图。初始化模块（Init）设计为一个 python 文件，主要负责对 Flask 提供的相关服务类如 Flask 框架的控制类、数据库控制类和会话控制类等等进行初始化，并对这些类的实例进行配置，最终提供给外部能正确操作会话层、数据库和 Flask 框架应用的接口。

配置模块（Config）设计为一个 python 文件，主要负责为初始化模块提供配置参数，以及为系统上传下载的文件存放路径、数据库资源定位和会话类型等

等。初始化模块能够从配置文件中导入配置信息。

启动模块（RunServer）设计为一个有着__main__函数的 python 文件，负责启动服务器，这样当双击或者其他模式下运行该文件，则服务器开启，系统可以正常使用，设计采用 gevent 的 WSGIServer，能够支持多线程，且服务比较稳定。

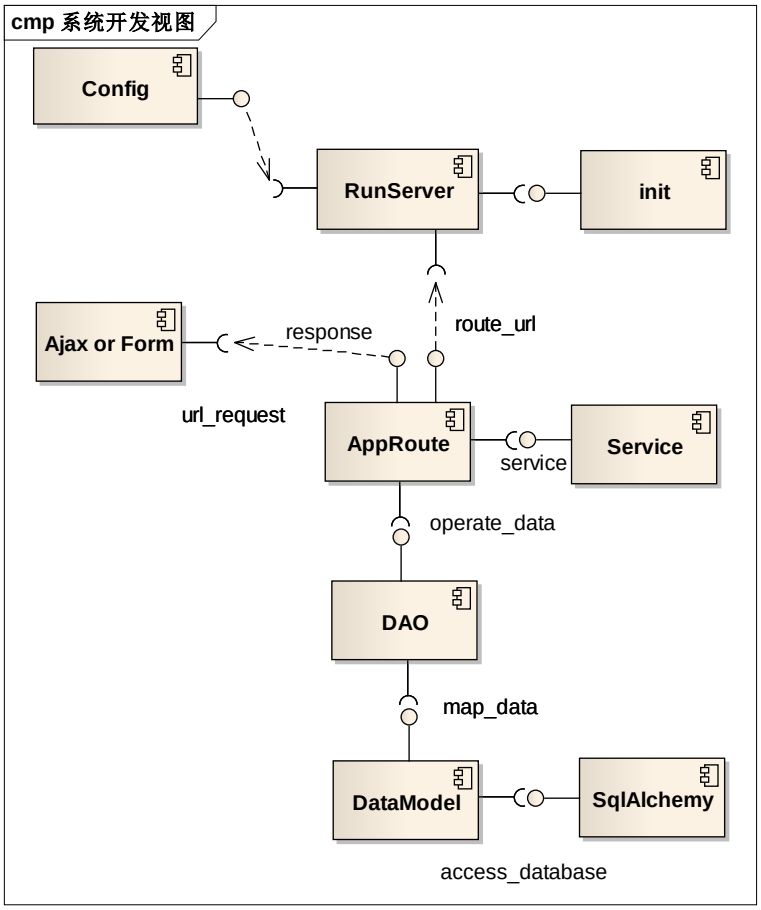

图 4.3 系统的开发视图

DAO 模块设计为 InvDao 类，UserDao 类，以及 HistoryDao 类，分别对应了对数据模型的 inventory, user, history 数据的访问，通过 DAO 模块可以为其他模块方便地提供访问和操作数据的接口，根据需求，InvDao 提供对物资进行增删改查、状态改变、请求、归还、报废、转让等操作的接口；UserDao 提供对用户进行增删查以及对用户进行身份验证的接口；HistoryDao 提供对历史记录的增和查的接口。

数据模型模块（DataModel）设计为 Inventory 类、User 类以及 History 类，

分别与数据库的 inventory, user, history 表所对应。Inventory 类负责定义物资的基本信息、状态、负责人和使用者等属性，User 类负责定义用户的基本信息、密码等属性，History 类负责定义历史记录的发生时间、操作人、操作的物资等属性。数据模型与数据库中的表相互映射，这是由 Flask SQLAlchemy 完成的。

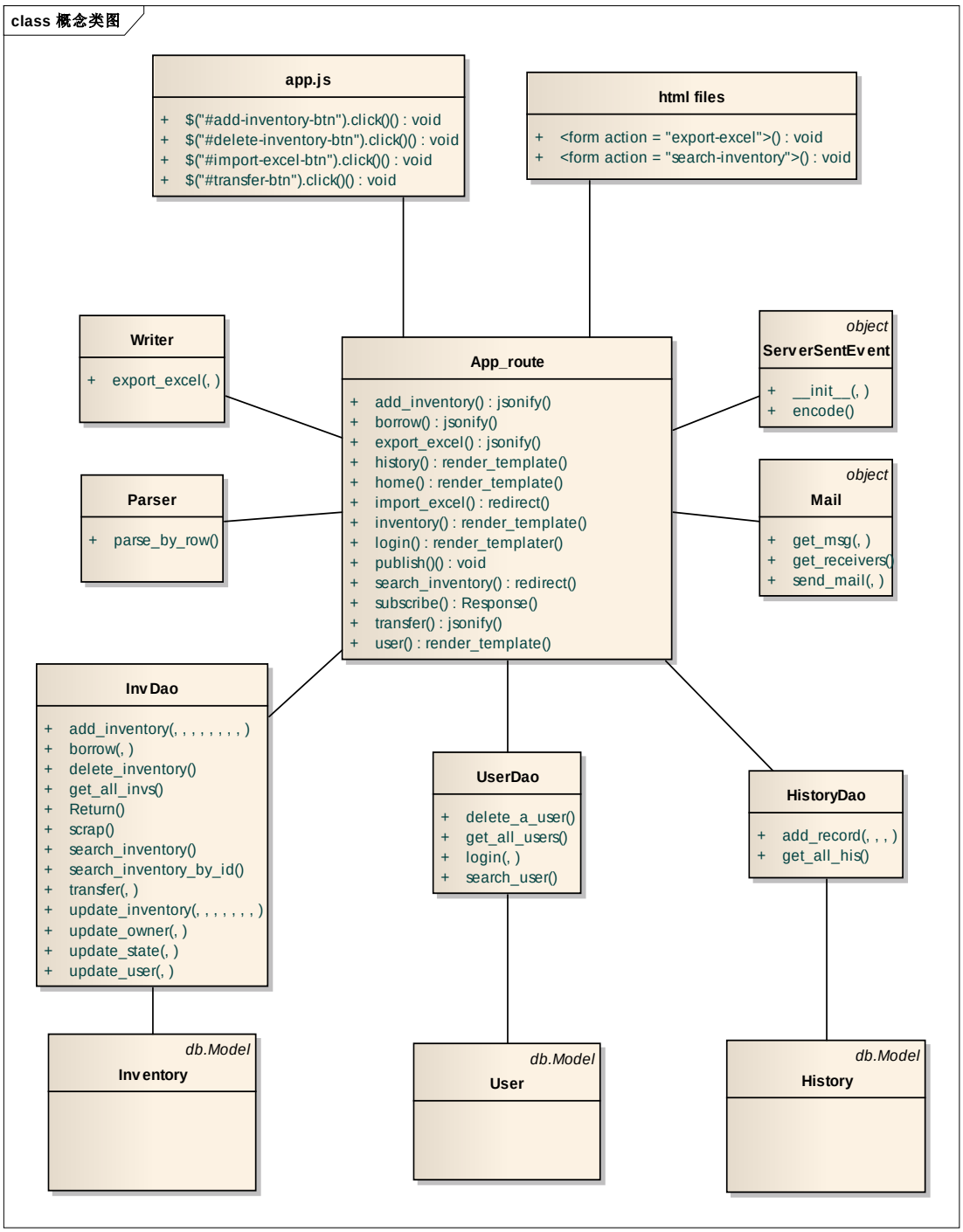

图 4.4 系统的概念类图

服务模块（Service）设计为 Writer 类、Parser 类、ServerSentEvent 类和 Mail 类。Writer 类负责导出 Excel 服务，能够从数据库中提取出物资信息并写入 Excel 表格。Parser 类负责导入 Excel 服务，能够从 Excel 表格中读取物资信息并创建添加物资到数据库。ServerSentEvent 类负责服务器推送消息协议的定义和编码。Mail 类负责邮件的制订和邮件的发送服务。

路由模块（AppRoute）设计为一个 python 文件，即 app_route.py，负责对浏览器发送的资源请求进行定位和响应，包括了页面的渲染、物资的操作、用户的操作、历史记录的查看等业务，以及部分服务器逻辑代码例如服务器推送消息和邮件通知。路由模块为服务器提供了路由 URL 的服务，所以在启动服务器模块中，必须导入路由模块，否则在浏览器进行访问时，服务器将无法找到相关资源来响应。

在如图 4.4 所示的系统的概念类图中，app.js 和 html files 是前端模块，通过 Ajax 或者 Form 表单请求服务器，App_route 对应的是路由模块，对所有的资源请求进行路由和控制，Writer、Parser、ServerSentEvent、Mail 分别是服务模块中的类，提供了相关的导入导出表格、在线消息通知和邮件通知的服务，再下来的三个 DAO 类是对应了 DAO 模块，提供了数据操作和访问的接口，最后，三个数据模型类对应了数据模型模块，分别是物资、用户和历史记录，对应了数据库中的表，关于数据库的设计将在下一节进行说明。

4.3 数据库设计

根据需求，物资管理系统所涉及到的实体有 inventory, user, history，其中 inventory 与 user 可以有多对一的关系，因此数据库设计为： inventory, user, history 三张表，而且 user 中有一个字段维护着该用户所负责的物资列表。

具体来说 inventory 表主要记录了物资管理中的标识信息，追踪记录，负责人、使用者以及当前状态，其字段及解释如表 4.1 所示。

表 4.1 inventory 表的设计

字段名称	字段类型	解释
id	Integer	主键，自增
tag	String(64)	物资的标签
name	String(64)	物资名称
PN	String(64)	Part Number
SN	String(64)	Serial Number
shipping	String(64)	物资的运送情况
capital	String(64)	属于购买还是自产
disposition	String(64)	安排情况
state	String(64)	状态
owner	String(64)	外键，该物资的负责人
user	String(64)	使用者

user 表主要记录了用户的邮箱，姓名，密码等信息，以及为了方便查询，维护一个自己所负责的物资列表，在 sqlalchemy 中称作一个 relationship，对于这一“字段”，不需要手动进行维护，sqlalchemy 有自动维护机制，当 inventory 表中增加了一条由某 owner 负责的物资记录时，该字段会自动增加对该物资记录。如果想知道用户负责的物资有哪些，则用 user.owned_invs.all()就可以得到所有物资记录。具体字段设计以及解释如下：

表 4.2 user 表的设计

字段名称	字段类型	解释
id	Integer	主键，自增
email	String(64)	用户邮箱，并设置为唯一性
username	String(64)	用户名称，并设置为唯一性
password_hash	String(64)	登录密码
owned_invs	relationship	负责的物资

history 表主要记录的是在系统中各项针对物资操作的历史记录，包括物资 ID，操作者，操作名称，时间日期，具体设计及解释如下表所示：

表 4.3 history 表的设计

字段名称	字段类型	解释
id	Integer	主键，自增
invid	Integer	物资 ID
username	String(64)	操作人的用户名
operation	String(64)	操作名称
time	String(64)	日期时间

4.4 模板设计

由于 Flask 使用的是 jinja2 模板引擎，所以在这一技术上，设计了一套相应的模板来方便前端开发实现。如图 4.5 所示，是多个模板间的继承关系。

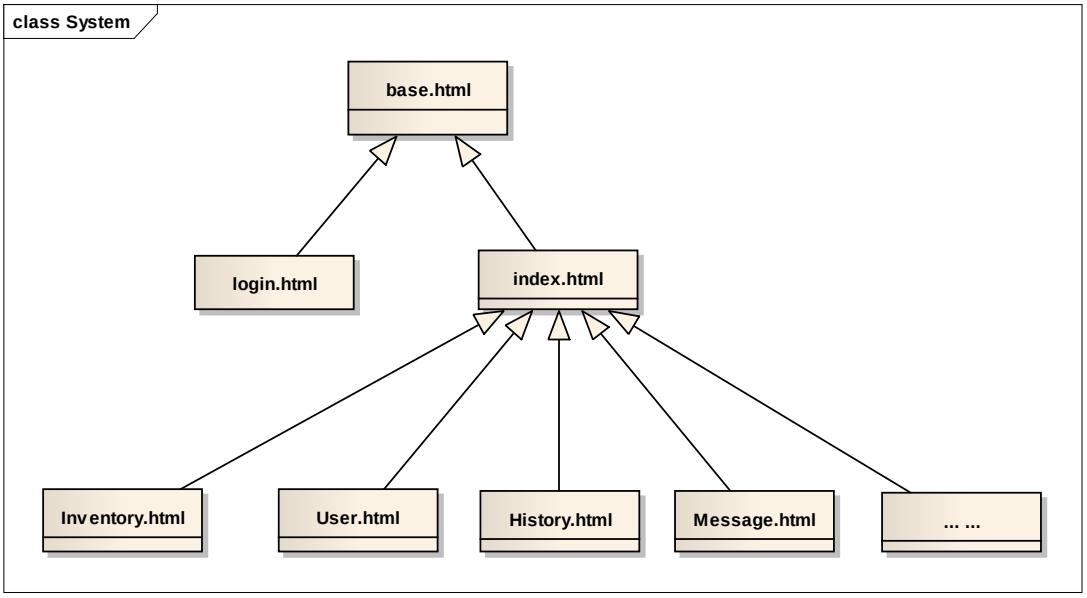

图 4.5 模板间的继承关系

其中，base.html 为最基础的模板，它定义了 html 文档，定义了<head>，把所有用到的 css，js 文件引入。只留下<body>的部分给后续的模板填充。

login 界面与登录后的界面布局差异大，所以，login 继承 base 后直接构成最终登录页面。

而其余的页面均有相似的布局和重复的部分，所以提出布局结构和重复的部分为 index.html，这样其他页面均继承 index，可以复用这部分代码。

所以在服务器端使用的渲染页面的语句 render_template('inventory.html')，返回的是经过 base.html 到 index.html 最后到 inventory.html 所渲染得出的页面，这项工作是由模板引擎实现的。

4.5 本章小结

本章主要介绍了物资管理系统的架构设计、模块设计、数据库设计以及前端开发所需要用到的模板设计。分层结构可以很好的满足 B/S 结构的设计和实现。模块化设计明确了各模块的分工与协作。数据库设计重点关注物资信息流动及信息变化。模板设计可以让前端页面代码更好地被组织和重用。

第五章 物资管理系统的实现

5.1 开发环境与系统依赖

本文的项目在 Eclipse Kepler IDE 上开发，为了支持 python，给 Eclipse 安装了 pydev 插件。物资管理系统所依赖的 flask 框架及 flask 插件和其他工具等则利用 pip install 进行了安装，需要安装的库有：flask，flask-sqlalchemy，flask-session，xlrd，xlwt 以及 gevent。其中，Python 采用 2.7 的版本，Flask 采用 0.10.1 版本，对应的 Jinja2 是 2.8 版本，Flask SqlAlchemy 采用 2.1 版本，Flask Session 采用 0.2.3 版本，xlrd 是 0.9.4 版本，xlwt 是 1.0.0 版本，gevent 采用 1.1.0 版本。前端用到的 Bootstrap 框架采用 3.3.5 版本，JQuery 是 2.1.3 版本。浏览器需要 IE10.0 及以上，以及 Chrome 浏览器 39 版本及以上和火狐浏览器 40 版本及以上。此外，本文系统是在 windows7 Service Pack1 的 64 位操作系统上开发、运行和测试的。

5.2 基于 Flask 框架的后台实现

5.2.1 初始化模块的实现

初始化模块负责 Flask 实例的创建、配置以及系统将要使用到的会话和数据库的创建。创建 Flask 实例只需要如图 5.2 中一条语句就能做到，而配置则设计成了从配置文件获取信息，所以将配置路径导入系统路径，再从文件中加载配置模块即可。会话和数据库的创建也分别利用已经创建好的 Flask 实例进行实例化。如图 5.1 所示：

```
app = Flask(__name__)
# Loading flask config.
sys.path.append(CONFIG_DIR)
app.config.from_object('config')
Session(app)
# Create DB instance.
db = SQLAlchemy(app)
```

图 5.1 初始化 Flask 实例的实现代码

5.2.2 配置模块的实现

配置模块设置了系统用到的常用变量，包括了配置路径、上传和下载 Excel 表格文件的存放路径、会话类型以及数据库资源定位。需要注意的一点是，配置文件是位于项目文件夹的一级目录下，所以取当前配置文件所在的目录路径为系统的配置路径，那么该文件夹中其他文件的定位就可以在配置路径的基础上用相对路径方式来表示。

```
import os
#path of the current file as the configure path
CONFIG_DIR = os.path.abspath(os.path.dirname(__file__))
#path setting of uploading and downloading excel file
UPLOAD_FOLDER = os.path.join(CONFIG_DIR, 'data', 'imported_files')
EXPORT_FOLDER = os.path.join(CONFIG_DIR, 'static', 'export_file')
#session type
SESSION_TYPE = 'sqlalchemy'
# DB setting.
SQLALCHEMY_DATABASE_URI = 'sqlite:///' + os.path.join(CONFIG_DIR, 'data',
'IM.db')
```

图 5.2 配置模块实现代码

5.2.3 启动模块的实现

启动模块导入 Flask 实例，从 gevent 导入 WSGIServer，利用 WSGIServer 的 serve_forever 启动 Flask，5000 是端口号，app 是导入的 Flask 实例，开发模式下可以将 app.debug 设置为 True，方便调试。多线程也设置为 True 的模式。如图 5.3 所示：

```
from IM import app
from gevent.wsgi import WSGIServer
#this is important for flask to find router, ignore the warning.
from IM.router import app_route
if __name__ == '__main__':
    app.debug = True
    app.threaded = True
    server = WSGIServer(("", 5000), app)
    server.serve_forever()
```

图 5.3 启动模块实现代码

需要强调的是，虽然在该模块中没有明显调用 app_route，有的 IDE 中会报出 warning，但是 from IM.router import app_route 这句话在这里不能少，因为 Flask 框架需要知道 app_route 的存在，在服务器启动之后，服务器需要用 app_route 来作 url 的解析和路由。

5.2.4 DAO 模块的实现

DAO(Data Access Object)模块，即数据访问模块，用来与底层数据库交互，InvDao 是与 Inventory 的数据操作相关，UserDao 是与 User 的数据操作相关。如图 5.4 所示，在 InvDao 中，根据系统的需求，实现了获取所有的物资列表，以及增、删、改的功能。

```
from IM import db
from model.Inventory import Inventory
import traceback

class InvDao():
    @staticmethod
    def get_all_invs():
        return Inventory.query.all()
    @staticmethod
    def add_inventory(tag, name, PN, SN, shipping, capital, disposition, status,
owner=''):
        '''need try catch exception or just check if unique'''
        inv = Inventory(tag, name, PN, SN, shipping, capital, disposition, status,
owner)
        try:
            db.session.add(inv)
            db.session.commit()
            return inv
        except:
            return None
    @staticmethod
    def delete_inventory(ids):
        …
    @staticmethod
    def update_inventory(id, tag, name, PN, SN, shipping, capital, disposition):
        …
```

图 5.4 InvDao 的实现代码

跟据需求，目前采用遍历各个字段的方式实现关键字查询，即在物资的每一个字段中进行关键字的查询，返回一个所有找到关键字的物资列表，具体实现代码如图 5.5 所示。由于数据库查询返回的结果是 list 结构，所以需要进行少量的代码工作来将各个 list 整合成一个 list 的结果返回。

```
def search_inventory(search_string):
    results=[]
    temp=[]
    try:
        results.append(Inventory.query.filter_by(tag = search_string).all())
        results.append(Inventory.query.filter_by(name = search_string).all())
        results.append(Inventory.query.filter_by(PN = search_string).all())
        results.append(Inventory.query.filter_by(SN = search_string).all())
        results.append(Inventory.query.filter_by(shipping = search_string).all())
        results.append(Inventory.query.filter_by(capital = search_string).all())
        results.append(Inventory.query.filter_by(disposition = search_string).all())
        results.append(Inventory.query.filter_by(status = search_string).all())
        results.append(Inventory.query.filter_by(owner = search_string).all())
    except Exception:
        traceback.print_exc()
    for result in results:
        if len(result) != 0:
            for item in result:
                temp.append(item)
    results = temp
    return results
```

图 5.5 查询物资的实现代码

另外的 UserDao 和 HistoryDao 的实现代码与 InvDao 的实现类似。不同的是，在 UserDao 中，对 User 的登录作了一个验证，验证方法调用的是 user 自身实现的验证密码的方法。

5.2.5 数据模型模块的实现

数据模型模块建立起与数据库表的一个映射。

如图 5.6 所示，用 db.Model 对自定义的 Inventory 类进行标识，用 db.Column 创建表的列，那么该 Inventory 类就对应了一个数据库表，初始化一个 Inventory 实例，就能通过 db.session.add(inv)来添加一条记录了。对__repr__重定义方便

直接打印物资信息，to_json 返回 json 表示。

```
class Inventory(db.Model):
    __searchable__ = ['tag', 'name']
    id = db.Column(db.Integer, primary_key = True, autoincrement=True)
    tag = db.Column(db.String(64), index=True)
    name = db.Column(db.String(64), index=True)
    PN = db.Column(db.String(64), index=True)
    SN = db.Column(db.String(64), unique=True, index=True)
    shipping = db.Column(db.String(64))
    capital = db.Column(db.String(64))
    disposition = db.Column(db.String(128))
    status = db.Column(db.String(64), index=True)
    owner = db.Column(db.String(64), db.ForeignKey('user.username'))
    user = db.relationship('User')
    def __init__(self, tag, name, pn, sn, ship, cap, dis, sta, owner=''):
        self.tag = tag
        …
    def __repr__(self):
        return '<Inventory %r:%r>' % (self.PN, self.SN)
    def to_json(self):
        return{
                'id':self.id,
                …
                }
```

图 5.6 Inventory 数据模型的实现代码

如图 5.7 所示，User 则映射了另一张表，verify_password 方法是在登录时用作密码的验证。

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key = True)
    #email, username, password…
    def __init__(self, username, email, password):
        self.username = username
        …
    def __repr__(self):
        return '<User %r>' % (self.username)
    def verify_password(self, password):
        if self.password_hash != password:
            return False
        else:
            return True
```

图 5.7 User 数据模型的实现代码

5.2.6 服务模块的实现

目前的服务模块中包含有对导入导出 Excel 表格的服务支持，有对SSE(server-sent-event)的服务支持，以及对邮件通知的支持。

如图 5.8 所示，导出 Excel 帮助系统建立起与 Excel 的连接，能够创建工作簿，创建表单，按照行列索引逐个填写数据，最后将表格保存到指定路径：

```
class Writer():
    def export_excel(data, file_path):
        workbook = xlwt.Workbook()
        sheet = workbook.add_sheet('sheet1')
        r_index = 0
        c_index = 0
        for row in data:
            for col in row:
                sheet.write(r_index, c_index, col)
                c_index += 1
            r_index += 1
            c_index = 0
        workbook.save(file_path)
```

图 5.8 Writer service 的实现代码

从 Excel 中导入数据是利用了 xlrd 工具，逐个从 Excel 表格中读取数据，读出数据随即利用 InvDao 把数据新建为一条记录并添加在表中。如图 5.9 所示：

```
import xlrd
from dao.InvDao import InvDao
from model.Inventory import Inventory

class Parser():
    @staticmethod
    def parse_by_row(datafile):
        workbook = xlrd.open_workbook(datafile)
        sheet = workbook.sheet_by_index(0)
        inv_args = []
        for row in range(sheet.nrows):
            for col in range(sheet.ncols):
                inv_args.append(sheet.cell_value(row, col))
            InvDao.add_inventory(inv_args[0], inv_args[1], inv_args[2], inv_args[3],
inv_args[4], inv_args[5], inv_args[6], inv_args[7])
            inv_args=[]
```

图 5.9 Parser service 的实现代码

SSE 中定义了协议所要求的数据格式以及方便信息格式化的方法：

```
# SSE "protocol"
class ServerSentEvent(object):
    def __init__(self, data):
        self.data = data
        self.event = None
        self.id = None
        self.desc_map = {
            self.data : "data",
            …
        }
    def encode(self):
        if not self.data:
            return ""
        lines = ["%s: %s" % (v, k)
                 for k, v in self.desc_map.iteritems() if k]
        return "%s\n\n" % "\n".join(lines)
```

图 5.10 SSE 协议接口的实现代码

5.2.7 路由模块的实现

路由模块的 app_route 的文件中实现了所有的跳转控制。导入了模板渲染、请求、响应、重定向、会话、jsonify，导入了 Flask 实例，从数据访问模块导入了相关类，从服务模块导入了相关类，以及其他的系统所需要的模块：

```
from flask import render_template, request, redirect, session, jsonify, Response, flash
from IM import app
from dao.UserDao import UserDao
from dao.InvDao import InvDao
from service.export_excel import Writer
from service.sse import ServerSentEvent
import os
import gevent
from gevent.queue import Queue
```

图 5.11 路由模块实现的依赖

实现 url 的解析和路由使用了 Flask 的@app.route()这个装饰符：

```
@app.route('/')
def welcome():
    return render_template('login.html')
```

图 5.12 Url 解析的实现代码

当在浏览器端访问如：http://127.0.0.1:5000/时，服务器会响应，调用该welcome 方法，则返回一个渲染的 login.html 页面给浏览器。

```
@app.route('/login', methods=['GET','POST'])
def login():
    if request.method == 'POST':
        name = request.form['login-name']
        password = request.form['login-password']
        if UserDao.login(name, password):
            session['username'] = name
            return redirect('/home')
        else:
            return render_template('login.html', error = 'wrong login name or
password')
    return render_template('login.html')
```

图 5.13 处理 POST 请求的实现代码

服务器端的逻辑大部分在该模块中，例如对登录的响应，当请求为 POST 类型时，从 request 获取 POST 表单中的 login-name 和 login-password 信息，然后利用 UserDao 的 login 方法进行登录验证，如果验证成功，那么设置 session['username']为当前用户，然后跳转至/home，也就是/home 所路由的方法中渲染的页面。否则，验证失败，就输出错误信息至浏览器。如果请求方式是 GET，那么只是渲染 login.html 页面给浏览器。具体实现如图 5.13 所示。

一些@app.route()是针对浏览器输入 URL 访问的，还有一些则只通过 POST 方式与服务器交互，通过表单中的 action 方法，js 文件中的 Ajax 方法或者其他一些监听事件的方法与之连接。如图 5.14 所示：

```
@app.route('/add-inventory', methods=['POST'])
def add_inventory():
    tag = request.form.get('tag')
    name = request.form.get('name')
    PN = request.form.get('PN')
    SN = request.form.get('SN')
    ship = request.form.get('ship')
    cap = request.form.get('cap')
    dis = request.form.get('dis')
    owner = request.form.get('owner')
    InvDao.add_inventory(tag, name, PN, SN, ship, cap, dis, "available", owner)
    return jsonify(result=True)
```

图 5.14 增加一条物资记录的服务器端实现代码

如图 5.15 所示，在 js 中实现这样一个将表单中的信息添加到 inventory 表中的操作，当用户填完数据，点击添加按钮时，触发事件，通过 js 向该方法发起 Ajax 请求，得到 json 格式的返回结果。

```
$('.new-btn').click(function(){
        $('#new-inventory').modal();
        $('#add-confirm').click(function(){
              $.ajax({
                    url: "http://127.0.0.1:5000/add-inventory",
                    type: "POST",
                    data: {tag: $('#inv-tag').val(),…
                    },
                    async: true,
               }).done(function(){location.reload();});
        });
   });
```

图 5.15 增加一条物资记录的前端实现代码

5.3 前端实现

5.3.1 前端模板

Flask 前端模板采用的是 Jinja2 模板引擎，在物资管理系统中，利用了模板引擎的继承特性，将共用的代码抽离出来，成为 base.html，再将<body>部分的共用代码抽离出来，成为 index.html，其他各个 html 文件继承该 index.html, 而 index.html 继承 base.html，另外有 login.html 继承 base.html。使用了 render_template 方法渲染具体的一个子模板时，模板引擎将子模板中继承的部分，有变量的部分，有函数的部分等计算出来，形成一个完整的 html 页面。

如图 5.16 所示，是登录页面，采用 Bootstrap 框架进行页面布局和界面元素美化，表单中的一项属性 action="{{ url_for('login') }}"就是连接到 app_route 的/login，从而向服务器发起登录的请求。url_for 是 Flask 的一个函数，这里它把 login 映射到 app_route 的/login，url_for 的参数也可以是一些相对路径，比如 url_for('static', filename='css/jquery-ui.min.css')，则是把路径指向了 static/css/jquery-ui.min.css。另外，使用了模板引擎提供的{% if %}{% endif %}方法，例如，判断是否有登录验证错误，如果有就把错误信息显示在页面。

```
{% extends "base.html" %}
{% block content %}
    <div class="container" align="center">
        …
        <div class="row">
            <div class="col-md-12" id="login-form" align="center">
                <form action="{{ url_for('login') }}" method="post">
                    <div id="div-login-name">
                        …
                    </div>
                    <div id='div-login-password'>
                        …
                    </div>
                    <input type="submit" value="Login" id = "login-btn">
                </form>
                {% if error %}<p class="error"><strong>Error:</strong> {{ error }}{%
endif %}
            </div>
        </div>
    </div>
{% endblock %}
```

图 5.16 使用了 jinja2 模板引擎的前端代码

{% extends "base.html" %}继承 base.html，{% block content %}{% endblock %}中间这一块代码就是名为 content 的 block，将被模板识别并填充到 base.html 声明 block content 的地方，也就是如图 5.17 所示中的<body>部分。

```
<html>
    <head id="header" role="banner">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        …
        <link href="{{url_for('static', filename='css/style.css')}}" rel='stylesheet'>
        …
        <script src="{{url_for('static', filename='js/jquery.js')}}"></script>
        …
    </head>
    <body>
        {% block content %}{% endblock %}
    </body>
</html>
```

图 5.17 base.html 实现可继承的页面模板代码

另外，前端采用 Bootstrap 实现，如图 5.18 所示，顶栏的导航组件在页面

向下滚动时是固定位置不变的，为了突出搜索物资设备这一高频率使用的功能，在主体风格为公司惯用的蓝色系之上，将 Search 按钮设计实现为红色，这样显得引人注目，让需要的用户很容易就能找到。页面总体风格比较简约干净，尤其在设计对物资操作的按钮栏上考虑到这一点，用隐藏需要触发的形式来减少页面的元素，减少对用户感官的压力，同时这遵循了所见即所得的设计原则，使得用户友好度增加。

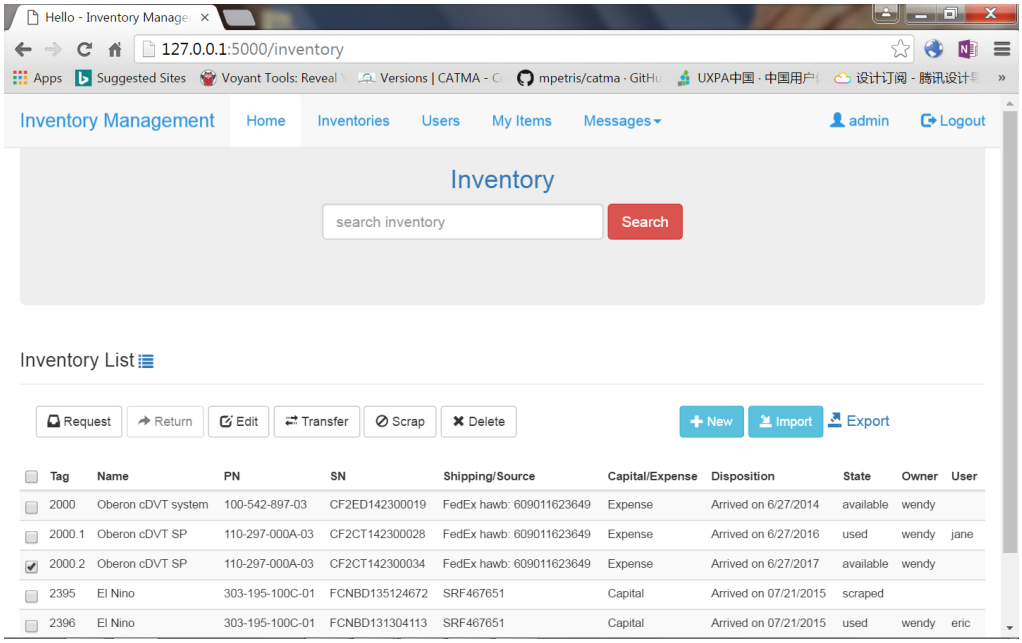

图 5.18 物资管理界面

5.3.2 交互细节

根据所见即所得原则，当页面上展示出一个物资的列表时，只有在用户点击勾选某一个物资时，其相应的操作按钮才出现，而当用户点击勾选了两个及以上时，编辑按钮成为不可用状态，其他按钮也会根据物资状态做出相应的可用或者不可用的变化，如图 5.19~5.21 所示：

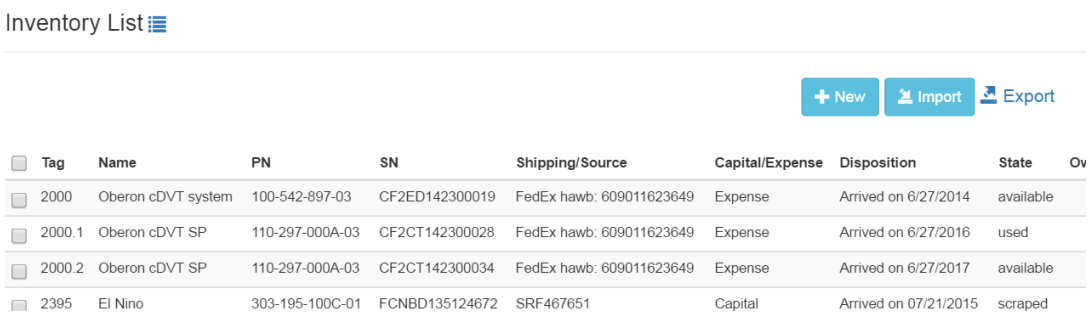

图 5.19 操作按钮的隐藏

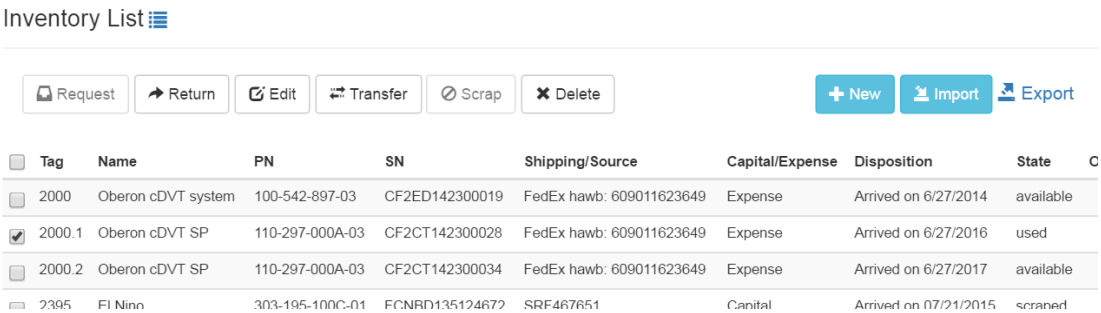

图 5.20 触发操作按钮

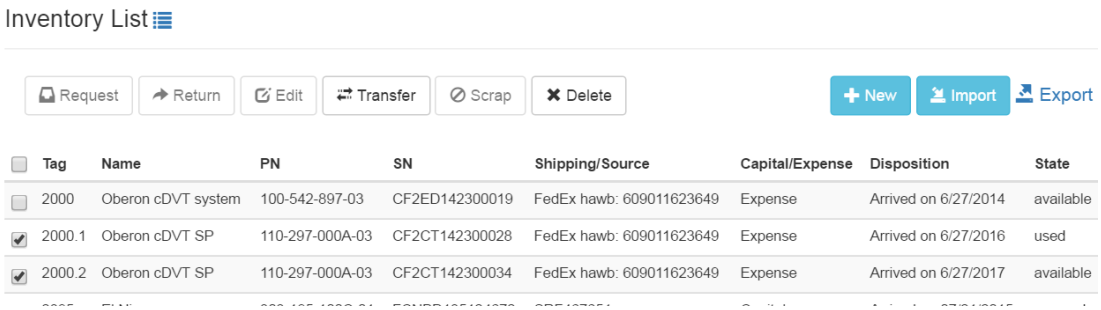

图 5.21 跟据物资状态给出可用的按钮

```
$('.check-item').click(function(){
        if ($('.check-item:checked').length == $('.check-item').length) {
                $('#check-head').prop('checked', true);
        }else {$('#check-head').prop('checked', false);}
        /*operation show and hide according to checked item*/
        if ($('.check-item:checked').length) {
              new operateBtn().operateBtn();
        }else{$('.operation').css('visibility', 'hidden'); }
});
```

图 5.22 复选框的逻辑代码

该交互细节用 JQuery 实现，当勾选框被点击时，通过调用 operateBtn 类的 operateBtn 方法来设置操作按钮为可见，并检查选中的数目及物资状态，对

操作按钮作出调整。如图 5.22 所示。

具体的对操作按钮进行变化处理的逻辑封装在 operateBtn 类中，operateBtn 是一个总的控制类，另外有 toggleBtn 类，是一个委托控制类，进一步封装因物资状态而变换操作按钮的逻辑，在 operateBtn 类中，除了调用 toggleBtn 类的服务，还实现了其他的控制细节，包括对操作盘的可见性的控制，检查当前用户的权限而对转让按钮做出相应的控制，对编辑按钮的控制，具体的代码实现如图 5.23 所示：

```
var operateBtn = function(){
    this.operateBtn = function(){
        new toggleBtn().toggleBack();
          $('.operation').css('visibility', 'visible');
          var trans_abled = true;
          $('.check-item:checked').each(function(){
              var currentRow = this.parentNode.parentNode;
              inv_state = currentRow.getElementsByTagName("td")[9].textContent;
              inv_owner = currentRow.getElementsByTagName("td")[10].textContent;

              if(inv_state != 'scraped' && current_user == 'admin' || current_user ==
inv_owner){
                  trans_abled = trans_abled & true;
              }else{
                  trans_abled = trans_abled & false;
              }
              if (trans_abled && inv_state!='scraped'){
                  $('.operation-transfer').prop('disabled', false);
              }else{
                  $('.operation-transfer').prop('disabled', true);
              }
              new toggleBtn().toggle(inv_state);
          });
          if ($('.check-item:checked').length > 1){
              $('.operation-edit').prop('disabled', true);
          }else{
              $('.operation-edit').prop('disabled', false);
          }
    }
}
```

图 5.23 复选框的逻辑代码之控制按钮可见性

其中，将每次跟据物资状态而变化按钮的逻辑以及恢复按钮状态的辅助性操

作的逻辑封装在 toggleBtn 类中，具体的代码实现如图 5.24 所示：

```
var toggleBtn = function (inv_state) {
    this.toggle = function(inv_state){
        switch (inv_state) {
        case 'available':
            $('.operation-return').prop('disabled', true);
            break;
        …
        }
    }
    this.toggleBack = function(){
        $('.operation-return').prop('disabled', false);
        $('.operation-borrow').prop('disabled', false);
        $('.operation-scrap').prop('disabled', false);
        $('.operation-transfer').prop('disabled', false);
    }

};
```

图 5.24 复选框的逻辑代码之跟据物资状态调整按钮

5.4 在线消息通知的实现

站内消息所通知的是即时消息，实现机制类似于发布/订阅(publish/subscribe)，原理是借助 SSE(Server-Sent Event) protocol，是由服务器向浏览器推送消息，见图 5.25：

```
def publish(inv_id, operation_msg):
    inv = InvDao.search_inventory_by_id(inv_id)
    current_user = session.get('username')
    users = 'admin'
    def notify():
        msg = str('The inventory named '+inv.name+' PN: '+inv.PN+' SN: '+inv.SN+
';'+ users+ ';'+current_user+';'+operation_msg)
        for sub in subscriptions[:]:
            sub.put(msg)
    gevent.spawn(notify)
```

图 5.25 服务器端消息推送代码

当产生了消息后，服务器端将利用 gevent.spawn 新建一个 greenlet 协程，把消息放至消息队列中，从而支持了并发消息。在浏览器端需要注册一个 eventsource 的 onmessage 事件，监听来自服务器端的消息，当消息队列有消

息的时候，会从服务器拿到包装成 SSE 格式的消息，过滤后，再把消息向相关的用户推送出去。浏览器端对消息的监听、过滤及显示，如图 5.26：

```
var evtSrc = new EventSource("/subscribe");
evtSrc.onmessage = function(e) {
    var data = e.data.split(';');
    var msg = data[0];
    var users = data[1];
    var current_user = data[2];
    var operation_msg = data[3];
    var user_list = users.split(' ');
    for (var i=0, len = user_list.length; i<len; i++){
        if(current_user == user_list[i]){
            eventOutputContainer.append("<div class='alert alert-success'>"+
                                        "<a href='#' class='close'
data-dismiss='alert' aria-label='close'>&times;</a>"+
                                        msg+operation_msg+
                                    "</div>");
        }
    }
};
```

图 5.26 浏览器端对消息的监听代码

被监听的/subscribe 资源，即服务器端的事件更新，如图 5.27：

```
@app.route("/subscribe")
def subscribe():
    def gen():
        q = Queue()
        subscriptions.append(q)
        try:
            while True:
                result = q.get()
                ev = ServerSentEvent(str(result))
                yield ev.encode()
        except GeneratorExit: # Or maybe use flask signals
            subscriptions.remove(q)
    return Response(gen(), mimetype="text/event-stream")
```

图 5.27 服务器端事件更新的代码

5.5 导入导出 Excel 表格的实现

5.5.1 导入 Excel

导入 Excel 的处理流程如下所述：

用户点击页面上的导入按钮，选择上传要导入的 Excel 文件，通过 html 表单发送给路由模块，相关的业务逻辑被调用执行：获取上传的文件，保存到指定目录下，再调用服务模块的 Parser 类来处理解析 Excel 文件并添加物资的事项。最后控制跳转返回到物资主页。可以看到更新后的页面已经把文件中的记录的物资添加至系统。

浏览器的代码实现如下：

```
<button type="button" class="btn btn-info btn-sm import-btn">
    <span class="glyphicon glyphicon-import"></span> Import
</button>
```

图 5.28 浏览器端的实现导入按钮的代码

Js 中触发一个 Bootstrap 的对话框事件：

```
$('.import-btn').click(function(){
        $('#import-file').modal();
});
```

图 5.29 导入 Excel 按钮触发事件的代码

后台的 python 实现如下：

```
@app.route('/import-excel', methods=['POST'])
def import_excel():
    ''' check file type == xlsx etc.
        if file exists, then server will overrides it by default.
    '''
    file = request.files['choose-excel-file']
    file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
    file.save(os.path.join(app.config['UPLOAD_FOLDER'], file.filename))
    Parser.parse_by_row(file_path)
    # the imported inventories will be added to the database and appended on the
page.
    return redirect('/inventory')
```

图 5.30 服务器端实现导入 Excel 表格的实现代码

5.5.2 导出 Excel

一个完整的响应流程如下所述：

用户点击页面上的导出按钮，浏览器端响应点击按钮事件，在响应处理函数中通过 Ajax 请求/export-excel 资源，服务器端响应，执行 app_route 中的 export_excel 方法，服务模块的 export_excel 服务被调用，创建了一个 xls 文件，并且将数据写入，保存到指定的 EXPORT_FOLDER 路径下，浏览器通过<a>的链接找到该文件，并通过 download 属性下载该文件。代码如图 5.31 所示。

```
$('#export-excel').click(function(){
        $.ajax({
                url: "http://127.0.0.1:5000/export-excel",
                type: "GET",
                async: false,
            });
  });
```

图 5.31 在 js 中通过 Ajax 发起导出 Excel 请求的代码

app_route 业务逻辑代码，如图 5.32 所示。其中，Writer.export_excel(data, file_path)的实现在前文的服务模块已经介绍过，可参见图 5.10 Writer service 的实现代码。

```
def export_excel():
    invs = InvDao.get_all_invs()
    data = []
    inv_args = []
    for inv in invs:
        inv_args.append(inv.tag)
        …
        data.append(inv_args)
        inv_args=[]
    print data
    file_path = os.path.join(app.config['EXPORT_FOLDER'], 'export.xls')
    print file_path
    Writer.export_excel(data, file_path)
    return jsonify(result=True)
```

图 5.32 服务器端实现导出 Excel 表格的代码

5.6 邮件通知的实现

邮件通知使用了 python 的标准库 smtplib。smtplib 包括一个 SMTP 类，可以创建 SMTP 的实例，通过实例的 sendmail 方法来实现发送邮件。[Hellmann, 2012] 在公司内部申请搭建了 SMTP 邮件服务器，而在测试时可以用 pip install dsmtpd 来安装一个本地测试用的 SMTP 邮件服务器，其默认端口号是 1025。由于 smtplib 是 python 的标准库，所以直接 import smtplib 即可。邮件发送通知的实现代码如图 5.33 所示。

```
import smtplib
sender = 'inventory_management@domain.com'
message = """From: From """+sender+"""
To: To #<to@todomain.com>
Subject: Inventory Management e-mail notice
This is a notice e-mail message from Inventory Management.
"""
class Mail(object):
    @staticmethod
    def send_mail(inv_id, msg):
        try:
            smtpObj = smtplib.SMTP('127.0.0.1', 1025)
            message = message.replace('#<to@todomain.com>', receiver)
            smtpObj.sendmail(sender, Mail.get_receivers(inv_id),   message+
get_msg(inv_id, msg))
            print "Successfully sent email"
        except Exception:
            …
    @staticmethod
    def get_receivers(inv_id):
        inv = InvDao.search_inventory_by_id(inv_id)
        receivers=[]
        owner_email = UserDao.search_user(inv.owner)
        user_email = UserDao.search_user(inv.user)
        receivers.append(owner_email)
        receivers.append(user_email)
        return receivers
    def get_msg(inv_id, msg):
        inv = InvDao.search_inventory_by_id(inv_id)
        inv_message = "The inventory PN: "+ inv.PN+…
        return inv_message+msg
```

图 5.33 邮件服务的实现代码

5.7 本章小结

本章详细描述了系统的各模块以及各技术点的关键实现，秉承了架构设计和模块设计的思路，在介绍关键的核心实现时附上了相关的代码，并且对系统开发过程中出现的一些细节问题的解决方案也做了阐述。

第六章 总结与展望

6.1 总结

本文综述了物资管理系统的需求、设计与实现，以及与物资管理系统相关的技术综述。物资管理系统对解决传统的通过 Excel 表格进行数据记录和管理的方式存在的弊端，以及对如何提高物资管理效率和减少出错率提供了解决方案：通过用户界面的操作提高效率，减少出错；通过导入导出 Excel 表格，兼容传统方式；通过在线消息通知和邮件通知等反馈方式，提高管理的时效性。

本文在需求阶段通过访谈与观察的方法获取了需求，通过面向对象的方法对需求进行了分析，给出了系统范围，并且通过用例图和用例描述的方法对需求进行了详细描述，这也帮助了与用户在需求方面达成一致理解。

本方案在设计和实现时遵循了三层结构的架构，注重了系统的模块性和系统的可拓展性，在前端和后台均下了功夫去设计和研发，在开发前端时，注重了代码的简洁及可重用，同时也注重了用户界面的友好性，在开发后台时，注重了系统的反馈和系统的可靠性。

此外，本文也通过物资管理系统的设计与实现呈现了如何实践 Flask 框架+前端的方式来实现一个 web 应用。同时也实践了利用模板引擎和前端框架使得前端开发变得简单优雅。本文的方案实现之后，取得了明显优化传统方式的效果，为工程师管理物资设备带来了方便，提高了管理物资的水平。

6.2 工作展望

在本文的工作中，反思物资管理系统，发现目前的物资管理系统还存在一些不足。较之以专业、成熟的管理信息系统，在支持管理者进行决策方面还做的不够。另外，如果能将基于角色控制访问的技术应用进来，就能支持更多样的权限访问控制。因此，从业务的角度来说，系统将会加入一些统计的业务，例如统计各项物资的使用频率，从而帮助管理员决策是否购买更多这样的物资作储备；统计各小组拥有物资的情况，来帮助管理员更好地平衡物资的分配。

从技术的角度来说，系统可以使用 js 库绘制统计图表，通过查询数据库获取数据进行统计，能够方便的支持统计业务的拓展。再者，由于当前对于管理员和普通用户的权限区别仅仅停留在 JavaScript 的判断层面上，随着业务的拓展，权限的变化，可以考虑使用 flask-rbac 基于角色的访问控制，从而区分出更多的角色，支持更多样化、自定义的角色访问权限。

参 考 文 献

[高凡等, 2015] 高凡，陈学卿，梁强，基于互联网的高校国资管理系统的设计，*计算机技术与发展*，2015, 25(10):174-178。

[刘肖, 2001] 刘肖，*基于Web的现代企业信息管理系统的研究和实现*，硕士论文，西北工业大学计算机应用，2001。

[马智亮, 2006] 马智亮，罗小春，李志新，基于万维网的工程项目管理系统综述，*土木工程学报*，2006， 39(10):117-120。

[叶锋, 2015] 叶锋，*Python 最新Web 编程框架Flask 研究*，电脑编程技巧与维护，2015。

[Cook, 2014] Darren Cook, *HTML5 数据推送应用开发,* 人民邮电出版社, 2014.

[Flask, 2016] http://flask.pocoo.org/, Flask framework introduction and community, maintained by Armin Ronacher, 2016.

[Frain, 2013] Ben Frain, *Responsive Web Design with HTML5 and CSS3,* Packt Publishing, 2013.

[Grinberg, 2015] Miguel Grinberg，*Flask Web 开发 基于Python 的Web 应用开发实战*，人民邮电出版社，2015.

[Hellmann, 2012] Doug Hellmann, *The Python Standard Library by Example,* Pearson Education, 2012.

[Jinja2, 2008] http://docs.jinkan.org/docs/jinja2/, Jinja2 template docs, maintained by Armin Ronacher, 2008.

[McLeod, 2001] Raymond McLeod, Jr. and George Schell, *Management Information systems, 8th Edition,* Pearson Education, 2001.

[Negrino, 2012] Tom Negrino and Dori Smith, *JavaScript: Visual QuickStart Guide,* Pearson Education, 2012.

[Noab, 2009] Gift Noab and Jones Jeremy, *Pytbon for UNIX and linux*

system administration, O'Reilly, 2009.

[Sommerville, 2010] Ian Sommerville, *Software Engineering, Ninth Edition,* Addison-Wesley, 2010.

[Spurlock, 2013] Jake Spurlock，*Bootstrap 用户手册*，人民邮电出版社，2013.

致　　谢

首先特别感谢任桐炜老师，在本文工作中给予我信任、帮助与鼓励，在论文开题及后续各个阶段都提供了耐心的指导和交流。

感谢公司部门的同事们，在开发项目中提出中肯的建议和讨论，帮助我拓展了思考和学习方式，学会了从不同角度理解看待问题。

感谢瑜伽老师们，通过细心的瑜伽指导，带我走向内在成长，帮助我舒缓压力，保持健康的身心状态从而高质量的完成工作。

感谢可可同学，始终以乐观积极的心态感染着我，让我成为更好的自己。

感谢身边的朋友们，你们的陪伴和关心，给予我勇气和力量。

感谢父母，没有你们的支持和爱，就不会有我的成长和蜕变。

版权及论文原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期：2016 年 5 月 25 日