基于Flask框架的物资管理系统的设计与实现

目录

摘要

第一章 引言

1.1项目背景

1.2国内外物资管理研究现状

1.3本文主要的研究工作

1.4本文的组织结构

第二章 技术综述

2.1办公自动化

2.2 工作流

2.2.1 工作流建模

2.2.2 基于Web的工作流

2.3 ERP

2.3.1 ERP简介

2.3.2 面向产品的ERP

2.4 Flask框架

2.4.1 Flask框架简介

2.4.2 Flask系统运作原理

2.4.3 Flask插件

2.5 JavaScript

2.6 JQuery

2.7 Bootstrap

2.8 开发环境与系统依赖

第三章：物资管理系统的需求分析

3.1物资管理系统的需求获取

3.2物资管理系统的涉众分析

3.3物资管理系统的需求列表与优先级

3.4 本章小结

第四章：物资管理系统的设计

4.1架构设计

4.2模块设计

4.3模式设计

4.4 数据库设计

4.5 界面设计

4.6 本章小结

第五章：物资管理系统的实现

5.1基于Flask框架的后台实现

5.1.1 初始模块的实现

5.1.2 配置模块的实现

5.1.3 启动模块的实现

5.1.4 DAO模块的实现

5.1.5 Data模块的实现

5.1.6 Model模块的实现

5.1.7 Service模块的实现

5.1.8 View跳转控制模块的实现

5.2 前端实现

5.2.1前端模板

5.2.2 交互细节

5.2.3 界面演示

5.3 站内消息通知的实现

5.4 邮件通知实现

5.4.1Flask-mail的使用

5.4.2 定制邮件模板

5.4.3 发送邮件

5.5基于python库xlwt 与xlrd的导入导出Excel的实现

5.5.1 Excel格式定制的分析

5.5.2 导入Excel

5.5.3 导出Excel

5.6 本章小结

第六章：总结与展望

致谢

第一章 引言

1.1项目背景

该项工作源于EMC公司内部需求，硬件维护工程师会对公司的众多硬件设备进行跟踪记录和管理，但是由于是用excel进行手动记录和管理工作，使得这项工作变得繁琐、容易出错而且容易丢失数据。因此，硬件维护工程师需要一个在线的管理平台来管理这些硬件设备。

物资管理系统旨在提高管理设备的效率，有效跟踪众多设备的状态，并且方便获得在线的消息和及时的邮件通知，能够进行对设备历史记录进行检索，能够自动化导入导出传统的excel表格数据，并根据各个team的不同要求定制excel表格。后续版本公司将考虑引进RFID，以实现对设备的高效跟踪和管理。

项目应用的前景在于实现公司内部物资管理的自动化，今后可能会接入RFID提高管理的效率。

1.2国内外物资管理研究现状

1.3本文主要的研究工作

物资管理系统是基于Flask框架的B/S系统，应用于公司内部跟踪和管理公司物资，该系统着力于改善传统的excel表格记录的手动方式，实现物资管理的自动化，建立消息通知和邮件通知的机制，让物资的状态变化及时通知到相关人员。笔者独立完成了整个系统的设计和实现，论文将具体介绍这些工作。

1. 需求分析

这一部分具体介绍物资管理系统的需求分析，需求获取方式是访谈+观察，列出了需求优先级，通过涉众分析了解到不同涉众的关注点，进行了快速的界面原型设计与其沟通，最终达成了一致；

1. 设计

介绍Inventory Management系统的设计，包括：

架构设计——数据层、业务逻辑层、表现层；

模块设计——dao模块，data模块，model模块，service模块，view模块，以及其他支持flask框架应用的配置模块、静态模板等；

模式设计——subscribe/publish模式用于消息通知；

数据库设计、界面交互设计；

1. 实现

介绍Inventory Management 系统的实现，包括：项目中遇到的技术难点和关键的解决方案，项目中是如何整合利用现有技术，给出部分关键实现代码，给出交互界面的演示；

第二章 技术综述

物资管理系统前端用HTML/CSS/JavaScript/JQuery以及Bootstrap框架技术，后台用python flask框架构建；站内消息用flask-redis，邮件通知使用flask-mail，excel导入导出使用python库xlwt和xlrd，系统使用B/S结构，架构设计为：数据层、业务逻辑层、表现层，模块设计为：dao模块，data模块，model模块，service模块，view模块，以及其他支持flask框架应用的配置模块、静态模板等。

通过在线搜索学习各项技术的使用，不停的试验应用到该项目中，对于各个技术点通过查找文献、技术文档等确定设计和实现方案，对于网站的架构遵循flask官网给出的建议，并参考了同事做的flask应用网站的架构。

2.1办公自动化

办公自动化出现在早几年前，现在这一模式已经被公司广泛应用。通过计算机的支持，定义办公流程，实现数据的存储、交换和管理，是办公自动化的核心理念。办公自动化的概念最早应用于打字机和复印机，它们都是把以前的手工劳动改造为机械劳动，现在作为一个术语不仅仅指把任务机械化，而且把信息能转变为电子格式加以应用。

2.2 工作流

工作流起源于生产组织和办公自动化领域，它将工作分解成任务、角色，按照定义的规则来执行任务并对任务进行监控，在有些手机上也有着类似软件：定义和执行工作流，将一些程序按照使用流程排列好，定义好彼此的数据接口，给一定的输入数据，得出最终的输出结果。现在有基于WEB的工作流系统，可以方便实现协同工作，实现办公的自动化。该项目中对Inventory的状态跟踪记录需要不同用户协作完成，而且Inventory的状态变化将会触发一些事件，利用工作流的机制，可以方便完成这个需求。

2.2.1 工作流建模

2.2.2 基于Web的工作流

2.3 ERP

企业ERP在业界已被广泛应用，其中面向产品的ERP拥有其优势，对公司拥有的inventory能够进行有效跟踪和管理。在Inventory Management中，一个关键的功能需求是对每一项设备进行详细的记录，包括购买、归档、使用、归还、报废、回收等等各个状态。

2.3.1 ERP简介

2.3.2 面向产品的ERP

2.4 Flask框架

Flask框架如今被业界认可，其特点在于灵活、轻量级，由python写成，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，官网上文档齐全，学习起来容易、灵活。

2.4.1 Flask框架简介

2.4.2 Flask系统运作原理

2.4.3 Flask插件

2.5 JavaScript

JavaScript近些年在实现网页前端方面有愈加广泛的应用，对构建动态的网页交互，实现前后端数据的交换起到重要作用，JavaScript结合HTML、CSS，共同实现灵巧易用的前端页面，使得网页开发更为简单方便，且拥有JQuery库的支持，能够更加快速构建一个动态页面。

2.6 JQuery

2.7 Bootstrap

Bootstrap是HTML、CSS、JavaScript的前端开发框架，使得实现一个响应式网站更加方便。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。为界面设计带来美观和协调。

2.8 开发环境与系统依赖

本文的项目是在Eclipse Kepler IDE上开发，为了支持python，给Eclipse安装了pydev插件，可以有两种安装方法：其一是下载pydev安装包，解压后将plugins和features包覆盖至Eclipse的文件夹下，重启Eclipse；其二是在线安装，在Eclipse的help->marketplace下搜索pydev，安装即可。

系统依赖的框架和工具等则利用pip install：

pip install flask

pip install flask-sqlalchemy

pip install flask-session

pip install xlrd, xlwt

pip install gevent

Flask-RBAC

An example of this might be a person who moved from the Accounts Payable (AP) department to the Accounts Receivable (AR) department.  Under the old model, the user gets additional security added so that they can perform the AR job functions, but the AP access is still in place.

### <http://searchmanufacturingerp.techtarget.com/feature/Before-implementing-ERP-understand-its-many-components>：ERP software for product-oriented companies

While the below components are still core to ERP, they cater more to companies with specific needs, such as product-oriented companies.

***Supply chain***

An ERP system that encompasses not only the company's internal operations, but the operations of [supply chain](http://searchmanufacturingerp.techtarget.com/definition/supply-chain-management) business partners and suppliers in the production of goods from raw materials, inventory and supplies gives companies much-needed visibility into their manufacturing processes.

***Distribution/warehousing***

ERP distribution and warehousing systems employ automation that enables the company's customer-facing sales force to link customer quotes and orders directly into back-office [inventory management](http://searchmanufacturingerp.techtarget.com/definition/Inventory-management), fulfillment and accounting systems. This ensures that orders are filled in a timely manner. Many ERP distribution systems also include comprehensive[warehouse management](http://searchmanufacturingerp.techtarget.com/definition/warehouse-management-system-WMS) functions that ensure that inventory in warehouses is optimized to meet the company's supply chain requirements.

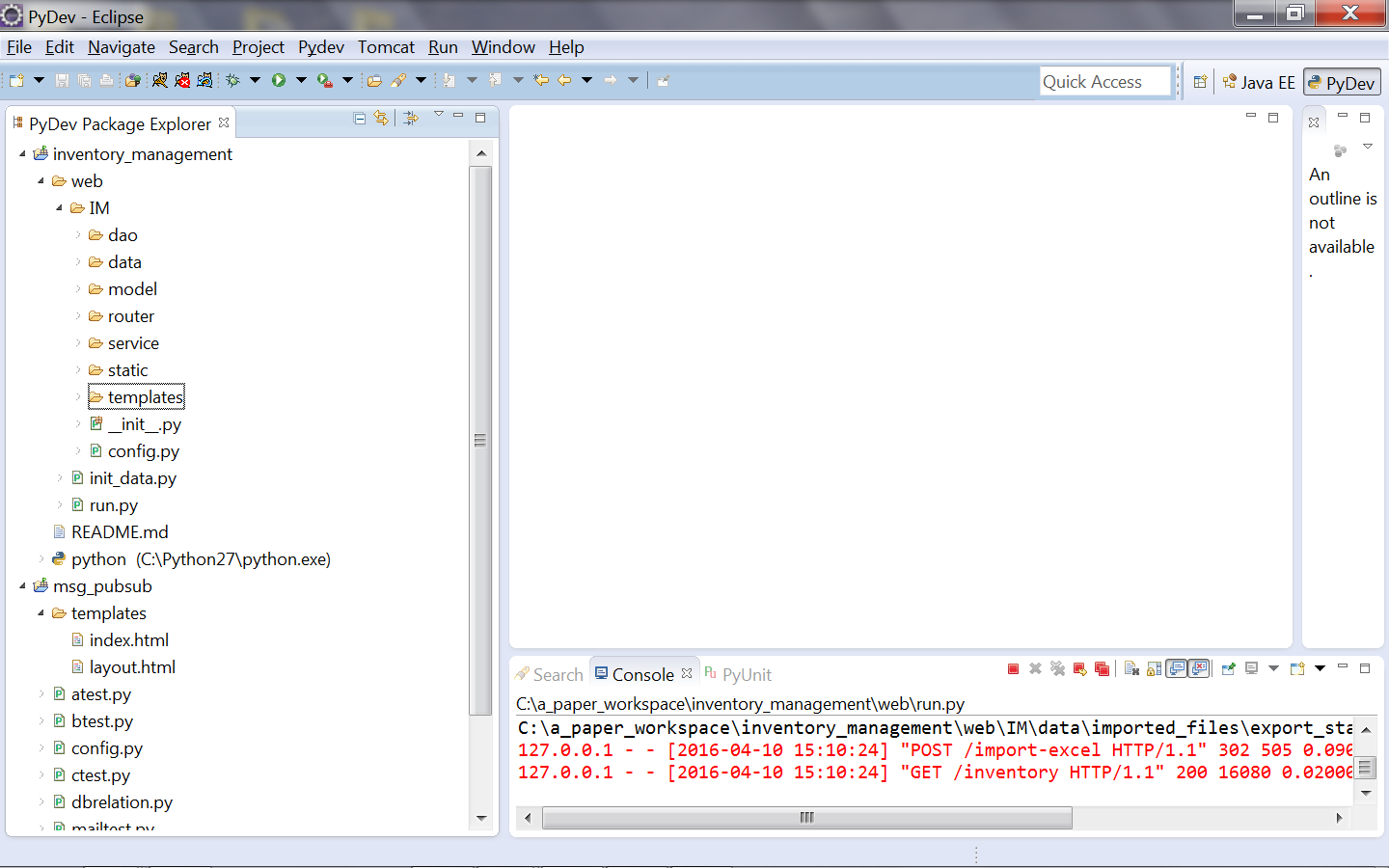
***Inventory***

An inventory management system optimizes inventory stocking and consumption and provides for both manual and automatic inventory forecasting. Companies can set order policies for individual parts and assemblies. The software also issues reports on inventory exception and potential oversupply conditions, and has the ability to track inventory across multiple locations.

第四章：物资管理系统的设计

4.1架构设计

4.2模块设计



4.3 数据库设计

数据库设计为： inventory, user, history 三张表

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，具体设计字段及解释如下：

|  |  |  |
| --- | --- | --- |
| 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inventory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表中增加了一条某某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，则直接用user.owned\_invs.all()就可以得到所有inventory记录。具体字段设计以及解释如下：

|  |  |  |
| --- | --- | --- |
| 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inventory |

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：

|  |  |  |
| --- | --- | --- |
| 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 |

4.3模板设计

由于Flask使用的是jinjia2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。如图所示，是多个模板间的继承关系。

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。

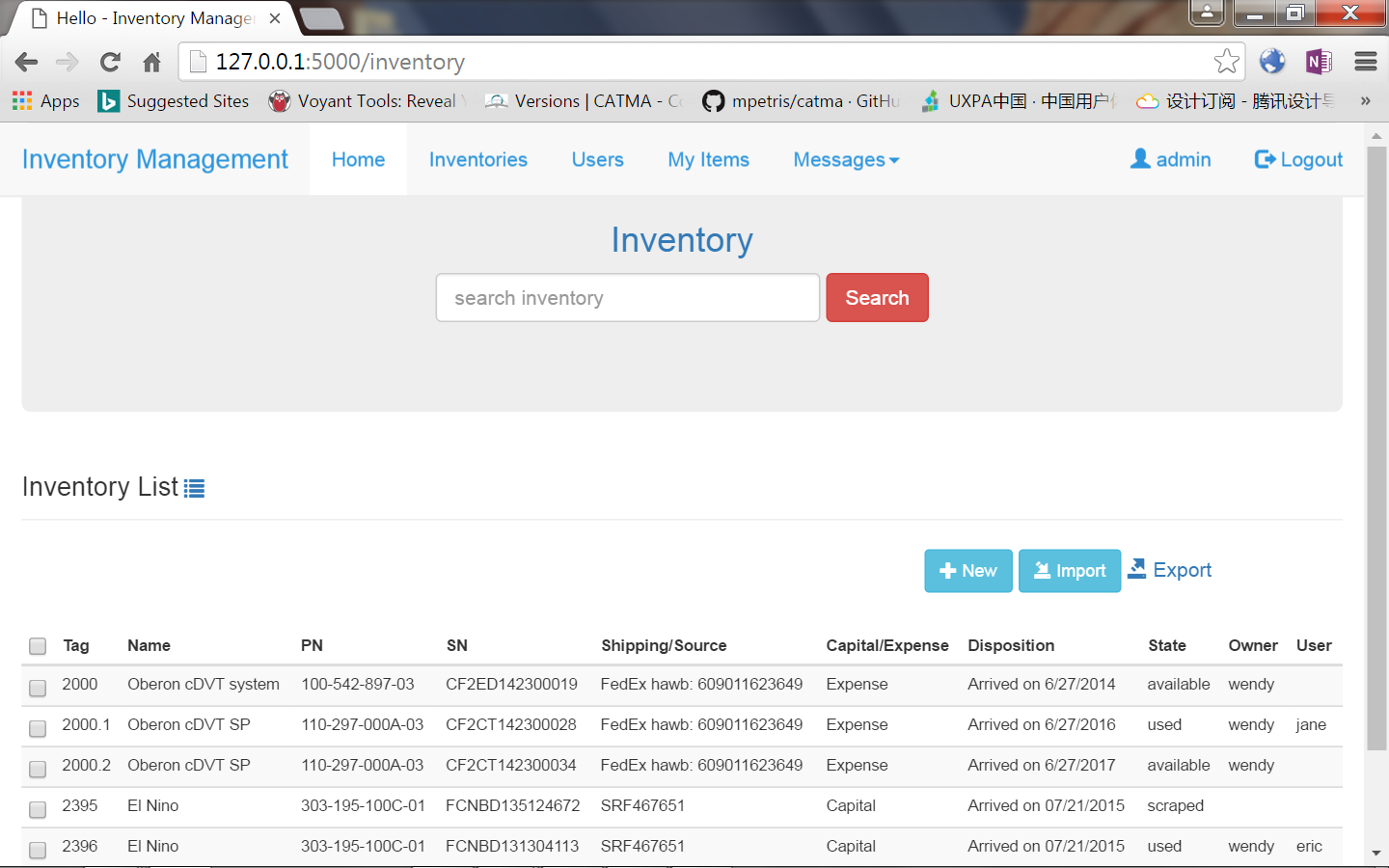
login界面与登录后的界面布局差异大，所以，login继承base后直接构成最终登录页面。

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。

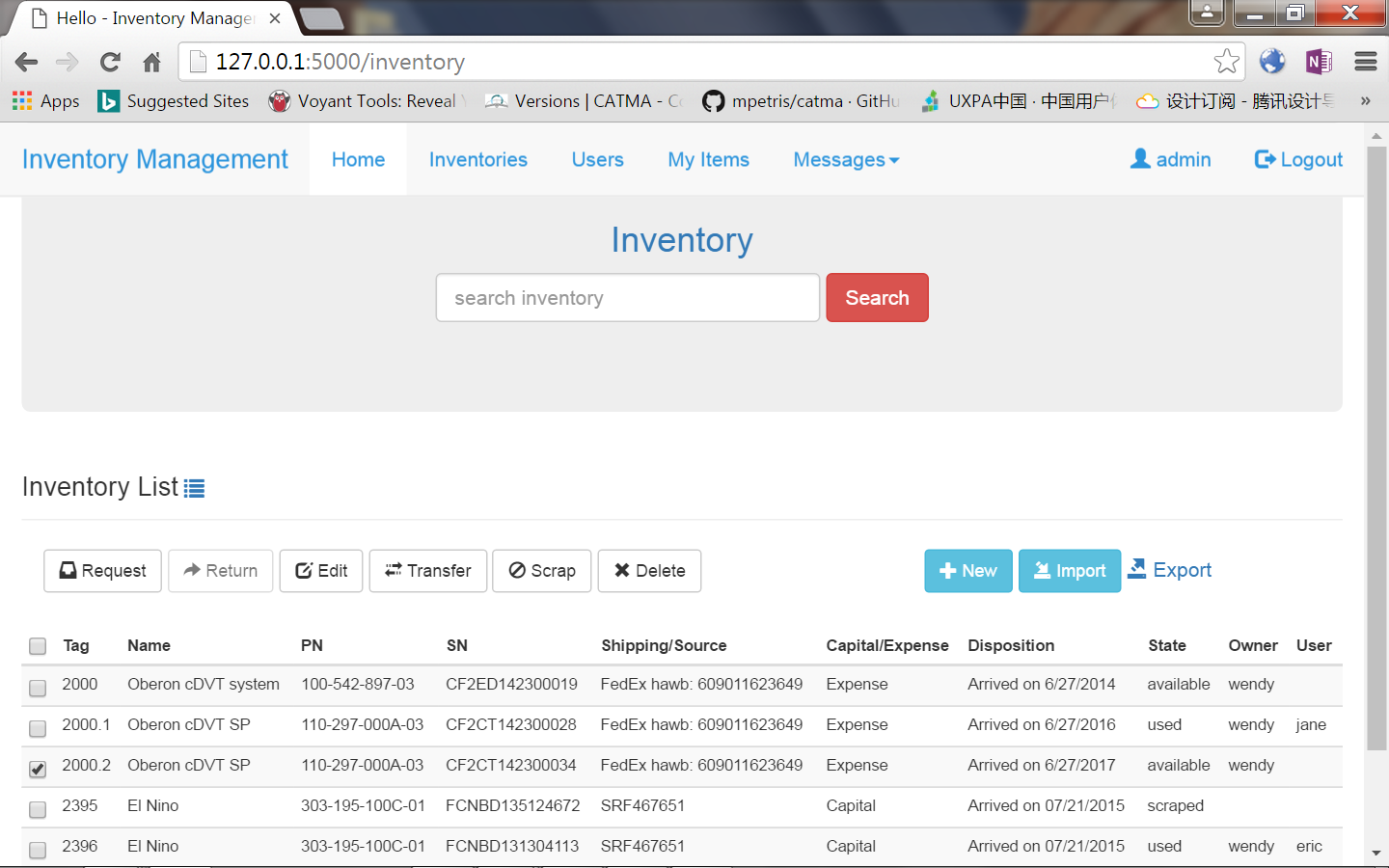
所以一个完整的inventory页面是从base渲染到index渲染最后到inventory渲染得出的页面，而在服务器端只需要一句，render\_template(‘inventory.html’)即可，模板引擎会自动找到继承的父模板，并把整个页面代码填充完整。

4.5 界面设计

采用bootstrap响应式网页设计，如图所示，顶栏的导航组件在页面下滑时固定其位置不变，突出search inventory这一高频率使用的功能，主体风格为公司的蓝色系，包括一些文字，按钮均设置成公司惯用的蓝色，而Search按钮用红色，这样显得引人注目，让需要的用户一眼就能找到。



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。如图所示，只有在用户勾选了某些inventory时，操作按钮组才会出现。



4.6 本章小结

第五章：物资管理系统的实现

5.1基于Flask框架的后台实现

5.1.1 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实例只需要如表格所示的一条语句就能做到，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。至此，整个初始化模块完成。

|  |
| --- |
| import os, sys  from flask import Flask  from flask.ext.sqlalchemy import SQLAlchemy  from flask.ext.session import Session  from config import CONFIG\_DIR  # Create Flask instance.  app = Flask(\_\_name\_\_)  # Loading flask config.  sys.path.append(CONFIG\_DIR)  app.config.from\_object('config')  Session(app)  # Create DB instance.  db = SQLAlchemy(app) |

5.1.2 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IM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') |

5.1.3 启动模块的实现

启动模块位于工程文件夹web下，与项目文件夹IM同属于一级。从IM中导入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。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。

|  |
| --- |
| 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.run(debug=True)  app.debug = True  app.threaded = True  server = WSGIServer(("", 5000), app)  server.serve\_forever() |

5.1.4 DAO模块的实现

DAO(Data Access Object)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。

如表格所示，在InvDao中，根据系统的需求，实现了获取所有Inventory，增、删、改、查Inventory，并且用一个简单遍历的方式实现了支持关键字查询。

|  |
| --- |
| 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):  invs=[]  for an\_id in ids:  invs.append(InvDao.search\_inventory\_by\_id(an\_id))  try:  for inv in invs:  db.session.delete(inv)  db.session.commit()  return True  except:  return False    @staticmethod  def update\_inventory(id, tag, name, PN, SN, shipping, capital, disposition):  inv = InvDao.search\_inventory\_by\_id(id)  inv.tag = tag  inv.name = name  inv.PN = PN  inv.SN = SN  inv.shipping = shipping  inv.capital = capital  inv.disposition = disposition  db.session.commit()    @staticmethod  def search\_inventory\_by\_id(search\_id):  return Inventory.query.filter\_by(id = search\_id).first()    @staticmethod  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 |

如表格所示，在UserDao中，对User的登录作了一个验证，验证方法是调用了user自身的一个验证密码的方法。

|  |
| --- |
| from IM import db  from model.User import User  class UserDao():    @staticmethod  def get\_all\_users():  return User.query.all()    @staticmethod  def delete\_a\_user(name):  return True    @staticmethod  def search\_user(name):  return User.query.filter\_by(username = name).first()    @staticmethod  def login(name, password):  user = UserDao.search\_user(name)  if user:  return user.verify\_password(password)  else:  return False |

5.1.5 Data模块的实现

Data模块用来存放db资源文件，以及上传的excel文件。关于db资源文件，是在init\_data文件中单独运行创建的。而且是为了初期测试系统使用的，如表格所示，用db.create\_all()创建出各个表，另外，创建了少量的inventory和user的记录，并且添加在表中用来测试：

|  |
| --- |
| from IM import db  from IM.model.User import User  from IM.model.Inventory import Inventory  db.create\_all()  #  inv1 = Inventory( '2000', 'Oberon cDVT system', '100-542-897-03', 'CF2ED142300019', 'FedEx hawb: 609011623649', 'Expense', 'Arrived on 6/27/2014', 'ok', 'wendy')  inv2 = Inventory( '2000.1', 'Oberon cDVT SP', '110-297-000A-03', 'CF2CT142300028', 'FedEx hawb: 609011623649', 'Expense', 'Arrived on 6/27/2016', 'ok', 'wendy')  inv3 = Inventory( '2000.2', 'Oberon cDVT SP', '110-297-000A-03', 'CF2CT142300034', 'FedEx hawb: 609011623649', 'Expense', 'Arrived on 6/27/2017', 'ok', 'wendy')  user1 = User('admin', 'admin@emc.com', 'admin')  user2 = User('wendy', 'wendy@emc.com', 'wendy')  db.session.add(inv1)  db.session.add(inv2)  db.session.add(inv3)  db.session.commit()  db.session.add(user1)  db.session.add(user2)  db.session.commit() |

5.1.6 Model模块的实现

Model是建立起与数据库表的一个映射。

如表格，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inventory信息，to\_json返回json表示。

|  |
| --- |
| from IM import db  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  self.name = name  self.PN = pn  self.SN = sn  self.shipping = ship  self.capital = cap  self.disposition = dis  self.status = sta  self.owner = owner    def \_\_repr\_\_(self):  return '<Inventory %r:%r>' % (self.PN, self.SN)    def to\_json(self):  return{  'id':self.id,  'SN':self.SN,  'PN':self.PN,  'tag':self.tag  } |

如表格，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。

|  |
| --- |
| from IM import db  class User(db.Model):  id = db.Column(db.Integer, primary\_key = True)  email = db.Column(db.String(64), unique=True, index=True)  username = db.Column(db.String(64), unique=True, index=True)  password\_hash = db.Column(db.String(128))  full\_name = db.Column(db.String(64), unique=False, index=True)  group = db.Column(db.Integer, unique=False, index=True)  active = db.Column(db.Integer, unique=False, index=True)    def \_\_init\_\_(self, username, email, password):  self.username = username  self.email = email  self.password\_hash = password    def \_\_repr\_\_(self):  return '<User %r>' % (self.username)  def verify\_password(self, password):  if self.password\_hash != password:  return False  else:  return True |

5.1.7 Service模块的实现

Service模块中有对导入导出excel表格的服务支持，有对server-sent-event的支持。

如表格所示，导出excel表格使用了xlwt的工具，该工具帮助系统建立起与excel的连接，能够创建工作簿，创建表单，并按照行列索引逐个填写表格数据，最后将写好的excel表格保存到指定路径：

|  |
| --- |
| import xlwt  class Writer():  @staticmethod  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)    #writer = Writer()  #data = [[0,1,2], [0,1,2],[0,1,2], [0,1,2,3]]  #file\_path = 'sheeet.xlsx'  #writer.export\_excel(data, file\_path) |

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Inventory记录并添加在表中。如表格所示：

|  |
| --- |
| 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))  print inv\_args  print 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=[] |

SSE中定义了协议所要求的数据格式：

|  |
| --- |
| # SSE "protocol"  class ServerSentEvent(object):  def \_\_init\_\_(self, data):  self.data = data  self.event = None  self.id = None  self.desc\_map = {  self.data : "data",  self.event : "event",  self.id : "id"  }  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) |

5.1.8 Router跳转控制模块的实现

app\_route该文件中实现了所有的跳转控制。从flask处导入了模板渲染、请求、响应、重定向、会话、jsonify，从IM导入了flask实例，从dao模块导入了各个Dao，从service模块导入了各个service，以及其他系统所需要的模块：

|  |
| --- |
| 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 |

对于实现url的解析和路由使用了flask的@app.route()这个decorator，例如最简单的welcome:

|  |
| --- |
| @app.route('/')  def welcome():  return render\_template('login.html') |

那么当在浏览器端访问像如：<http://127.0.0.1:5000/>时，那么服务器会响应，调用welcome方法，即返回一个渲染了模板的login.html给浏览器。

很多服务器逻辑都在该模块中，例如登录的响应，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，那么设置好session[‘username’]为当前用户，然后跳转至/home，也就是/home下渲染的页面。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。具体实现如表格所示：

|  |
| --- |
| @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') |

有一些@app.route()是浏览器输入访问的，还有一些是只通过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。例如add\_inventory，见表格：

|  |
| --- |
| @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')  #status = request.form.get('status')  owner = request.form.get('owner')  InvDao.add\_inventory(tag, name, PN, SN, ship, cap, dis, "ok", owner)  return jsonify(result=True) |

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add按钮时，触发事件，通过js向该方法发起ajax请求，得到jsonify的返回结果。js中的实现如表格所示：

|  |
| --- |
| $('.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(),  name: $('#inv-name').val(),  PN: $('#inv-PN').val(),  SN: $('#inv-SN').val(),  ship: $('#inv-shipping').val(),  cap: $('#inv-capital').val(),  dis: $('#inv-disposition').val(),  owner: $('#inv-owner').val(),  },  async: true,  }).done(function(){  location.reload();  });  });  }); |

5.2 前端实现

5.2.1前端模板

根据flask框架的约定，前端代码的html模板放入到templates目录下, css, js, img, fonts等其他文件放入到static目录下，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，将body部分的共用代码再抽离出来，成为index.html，其他各个html继承该index.html, 而index.html继承base.html，另外有login.html继承base.html。在使用render\_template方法渲染具体的一个子模板时，模板引擎会将子模板中继承的部分，有变量的部分，有flask函数的部分等等全都运行计算出来，形成一个完整的html页面。

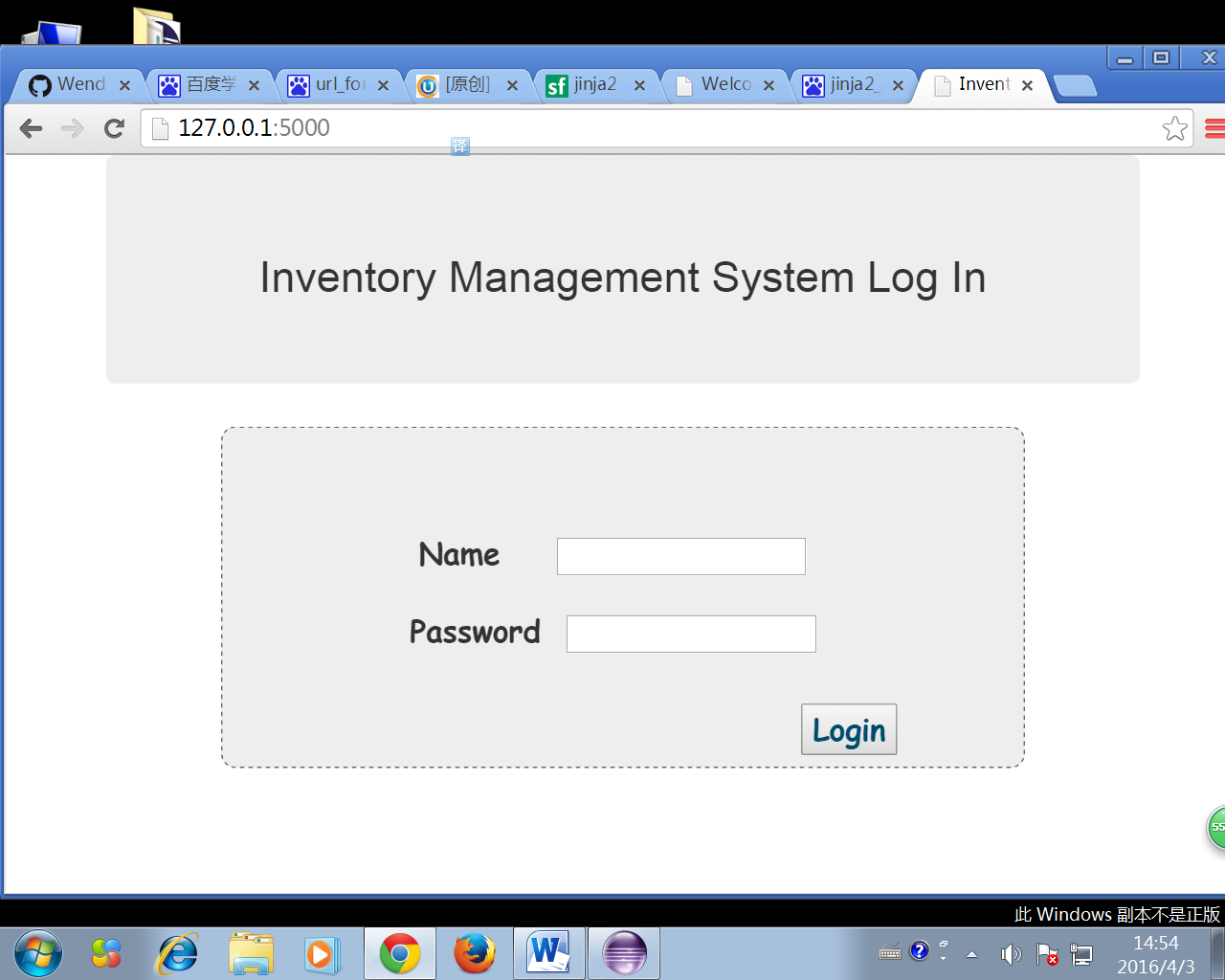
如表格所示，是登录页面，采用bootstrap框架进行页面布局和界面元素美化，表单中的一项属性action="{{ url\_for('login') }}"就是连接到app\_route的/login，从而向服务器发起登录的请求。url\_for是flask的一个函数，这里它将把login映射到app\_route的/login，url\_for的参数也可以是一些相对路径，比如url\_for('static', filename='css/jquery-ui.min.css')，则是把路径指向了static/css/jquery-ui.min.css。另外，使用了模板引擎提供的{% if %}{%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

|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <div class="container" align="center">  <div class="jumbotron" align="center">  <h2> Inventory Management System Log In </h2>  </div>    <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">  <p id="label-name">Name </p>  <input type="text" name="login-name" id="login-name">  </div>  <div id='div-login-password'>  <p id="label-password">Password</p>  <input type="password" name="login-password" id="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 %} |

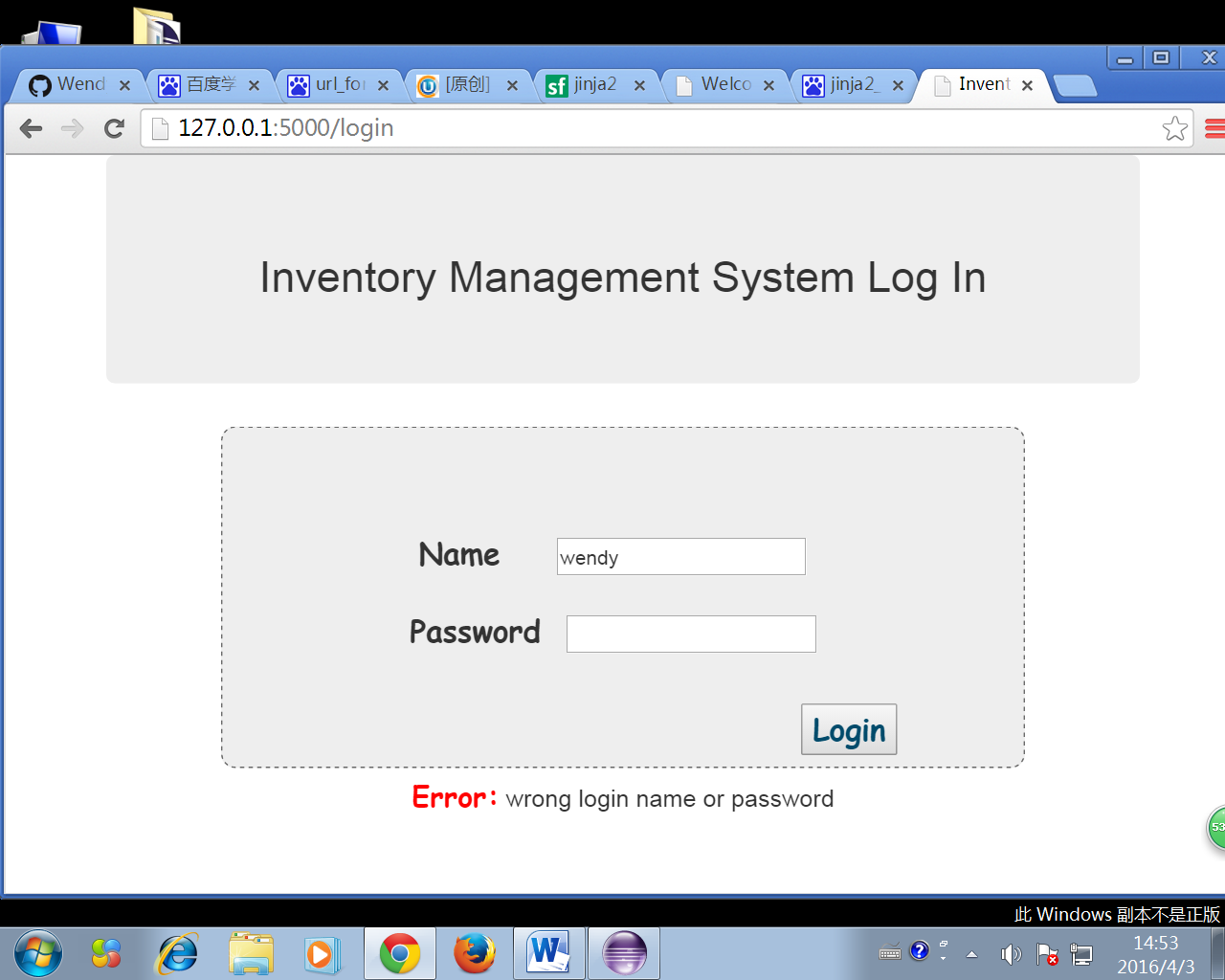
{% extends "base.html" %}是继承base.html的意思，{% block content %}{% endblock %}中间这一块代码就是名为content的block，将被插入到base.html声明block content的地方，如表格所示，也就是<body>{% block content %}{% endblock %}</body>这里。

|  |
| --- |
| <html>  <head id="header" role="banner">  <meta name="viewport" content="width=device-width, initial-scale=1">  {% if title %}  <title>{{ title }} - Inventory Management</title>  {% else %}  <title>Inventory Management</title>  {% endif %}  <link href="{{url\_for('static', filename='css/style.css')}}" rel='stylesheet'>  <link href="{{url\_for('static', filename='css/login.css')}}" rel='stylesheet'>  <link rel="stylesheet" href="{{url\_for('static', filename='css/bootstrap.min.css')}}">  <link rel="stylesheet" href="{{url\_for('static', filename='css/jquery-ui.min.css')}}"></script>    <script src="{{url\_for('static', filename='js/jquery.js')}}"></script>  <script src="{{url\_for('static', filename='js/jquery-ui.min.js')}}"></script>  <script src="{{url\_for('static', filename='js/bootstrap.min.js')}}"></script>  <!--the order is important for the app.js is based on jquery -->  <script src="{{url\_for('static', filename='js/app.js')}}"></script>    <script type="text/javascript">  var $SCRIPT\_ROOT = {{ request.script\_root|tojson|safe }};  </script>  </head>  <body>  {% block content %}{% endblock %}  </body>  </html> |

最终的登录页面如图所示：



登录出错时的提示，如图所示：



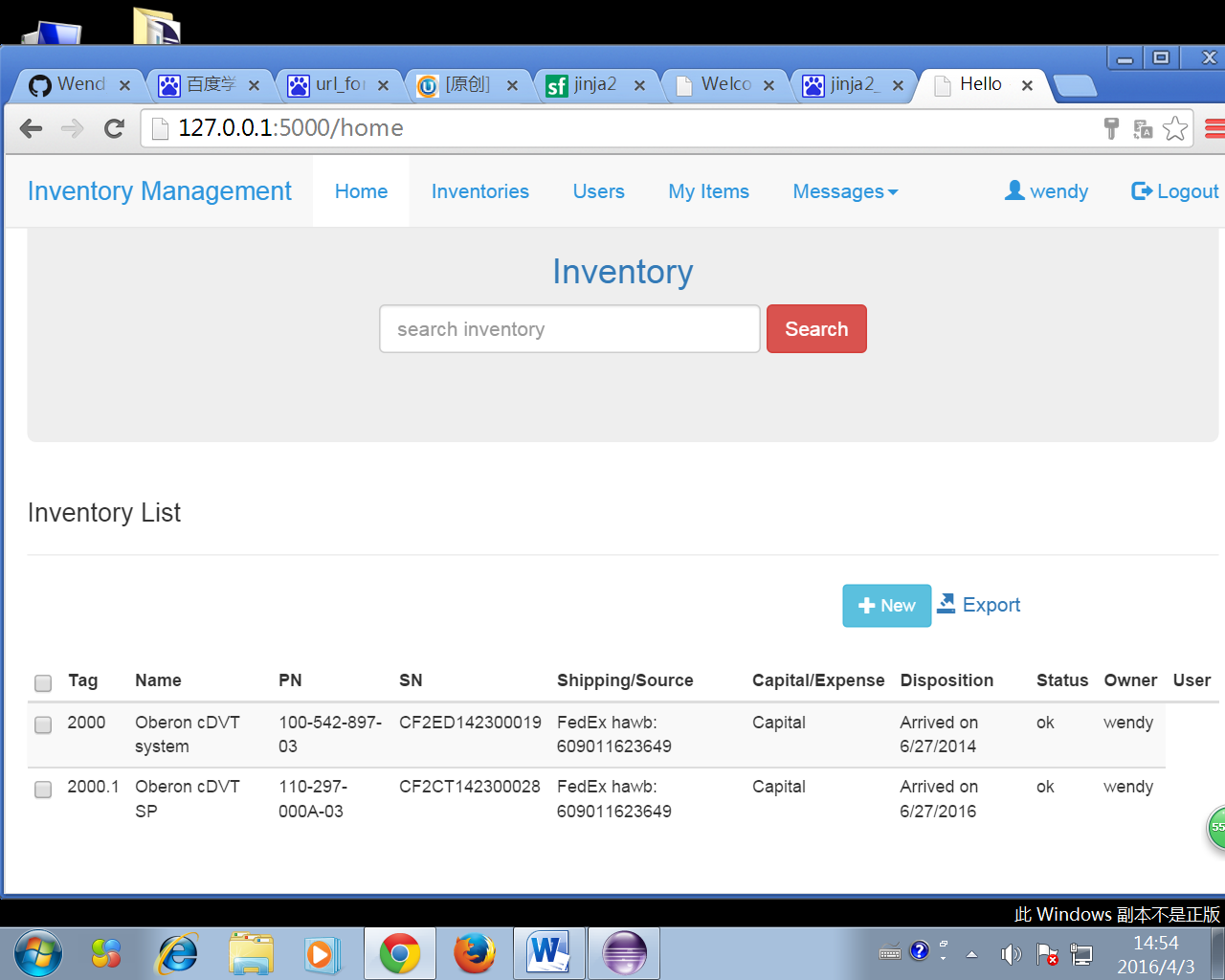
同样的，index.html继承base.html, index.html中的block content 部分被插入到base.html的<body></body>中。

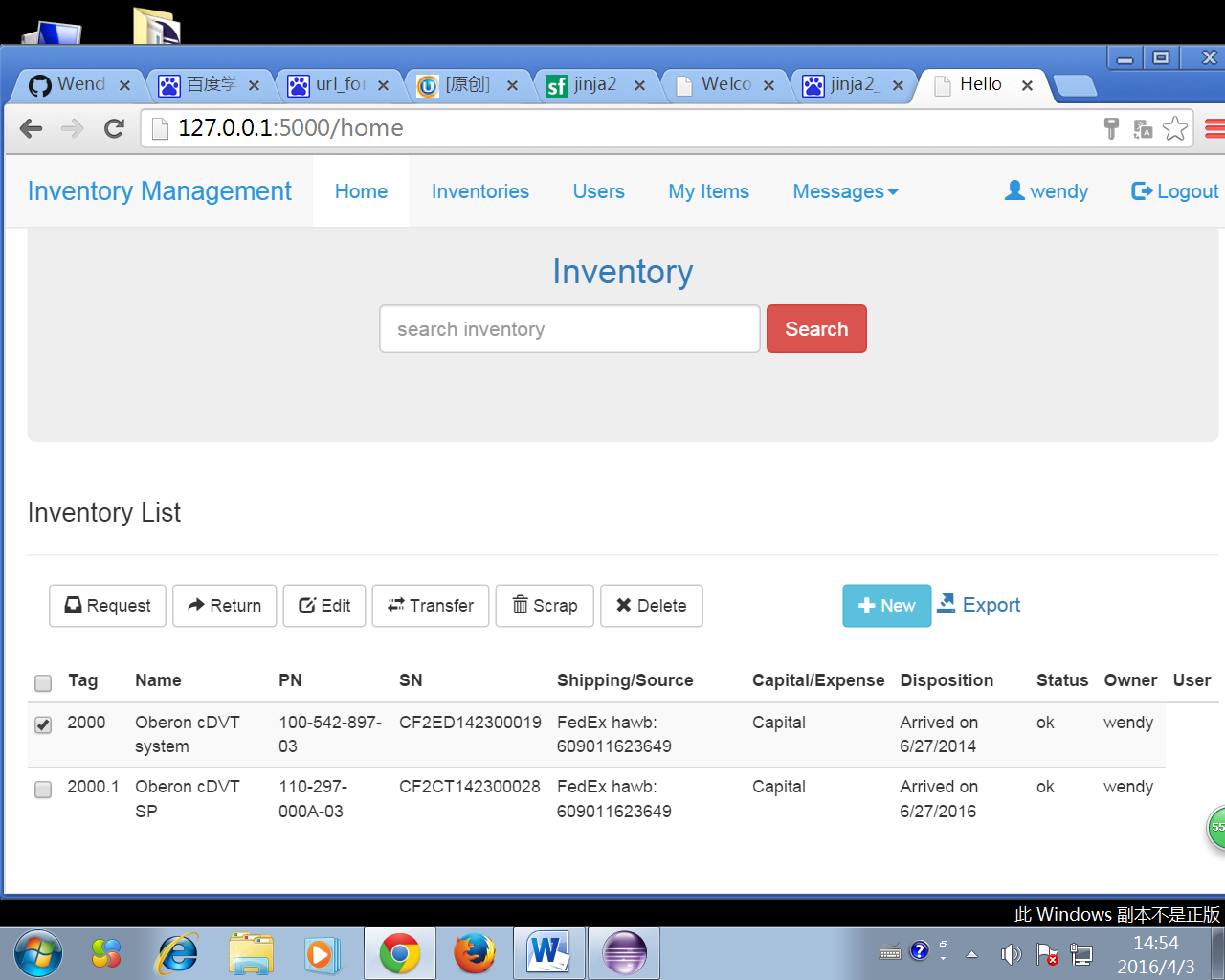
|  |
| --- |
| {% extends "base.html" %}  {% block content %}  <nav class="navbar navbar-default navbar-fixed-top">  <div class="container-fluid">    <div class="navbar-header">  <a class="navbar-brand" href="#">Inventory Management</a>  </div>    <ul class="nav navbar-nav">  <li class="active"><a href="{{url\_for('home')}}">Home</a></li>  <li><a href="{{url\_for('inventory')}}">Inventories</a></li>  <li><a href="#">Users</a></li>  <li><a href="#">My Items</a></li>  <li class="dropdown">  <a class="dropdown-toggle" data-toggle="dropdown" href="#">Messages<span class="caret"></span>  </a>  <ul class="dropdown-menu">  <li><a href="#">Borrow request</a></li>  <li><a href="#">Transfer request</a></li>  <li><a href="#">Return request</a></li>  <li><a href="#">My borrows</a></li>  <li><a href="#">My transfers</a></li>  <li><a href="#">My returns</a></li>  </ul>  </li>  </ul>    <ul class="nav navbar-nav navbar-right">  <li><a href="#" id="username"><span class="glyphicon glyphicon-user"></span> {{name}}</a></li>  <li><a href="/login"><span class="glyphicon glyphicon-log-out"></span> Logout</a></li>  </ul>    </div>  </nav>  <div>  {% if task == 'inventory' %}  {% block showcontent %}{% endblock %}  {% endif %}  </div>  {% endblock %} |

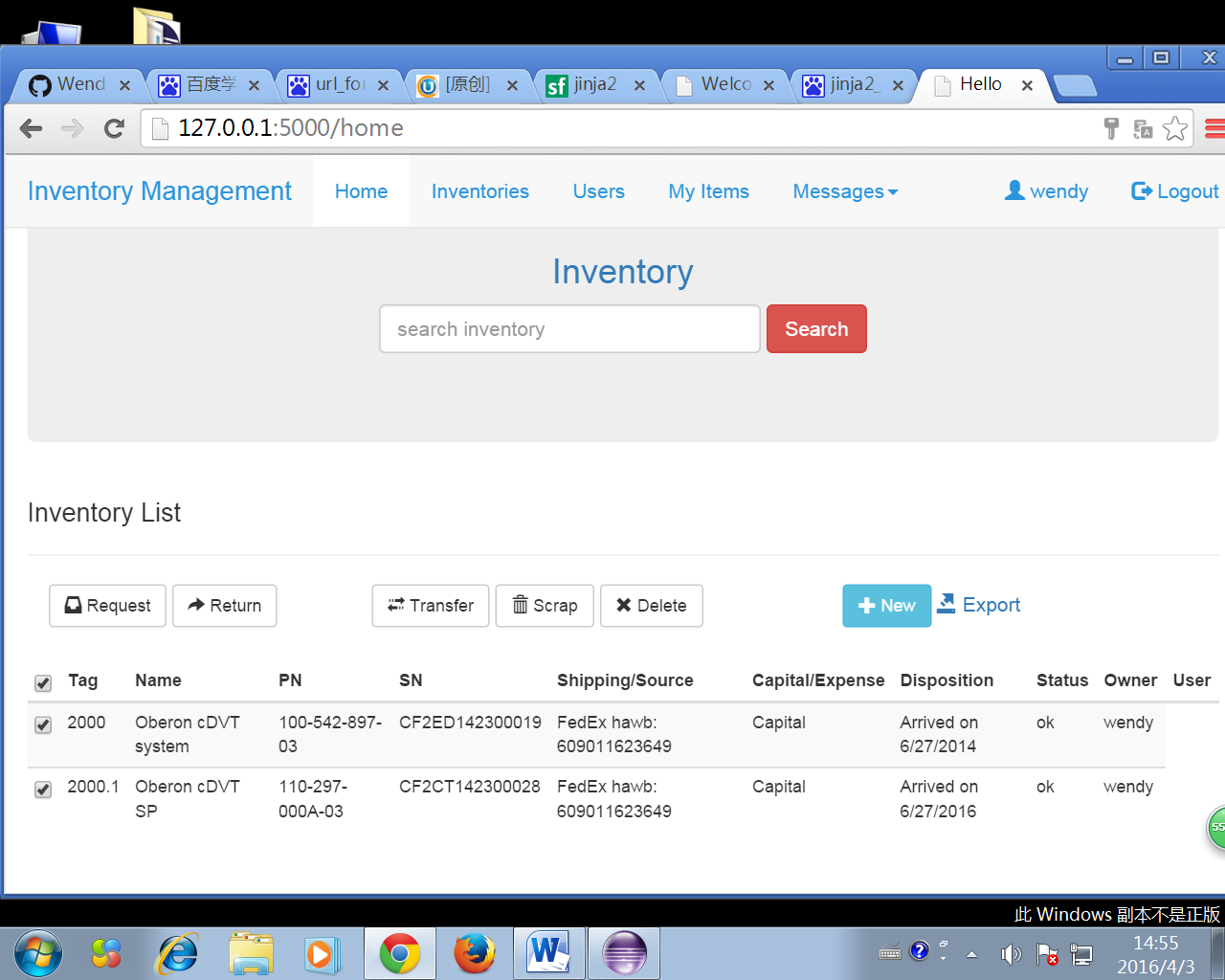
5.2.2 交互细节

一组checkbox的使用：

根据所见即所得原则，页面上展示出一个inventory的列表，只有当用户点击勾选某一个inventory时，其相应的操作盘(operation panel)才出现，而当用户点击勾选了两个及以上时，操作盘中的edit操作消失，因为edit操作只是针对一个inventory进行的，两个以上没有意义，所以就将其隐去，如图所示：







该交互细节的实现是用js，当check-item被点击时，设置operation为可见，检查check的数目，大于1则将operation-edit按钮隐去。其余代码是处理check-item与check-head的相互关系，要保证check-head表现出的行为是勾选了所有项。如表格所示：

|  |
| --- |
| /\* check items \*/  $('.check-item').click(function(){  /\*check-head change according to check-item\*/  if ($('.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) {  $('.operation').css('visibility', 'visible');  if ($('.check-item:checked').length > 1){  $('.operation-edit').css('visibility', 'hidden');  }  }else{  $('.operation').css('visibility', 'hidden');  }  });    /\* check-head\*/  $('#check-head').change(function(){  $('.check-item').prop('checked', $(this).prop('checked'));  if (this.checked == true){  $('.operation').css('visibility', 'visible');  if ($('.check-item:checked').length > 1){  $('.operation-edit').css('visibility', 'hidden');  }  }else{  $('.operation').css('visibility', 'hidden');  }  }); |

5.2.3 界面演示

5.3 站内消息通知的实现

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，由服务器向浏览器推送消息，当有消息时，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，作一番过滤后，把消息向相关的用户推送出去。

服务器端消息的产生及推送，见表格：

|  |
| --- |
| 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) |

浏览器端对消息的监听、过滤及显示，见表格：

|  |
| --- |
| 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>");  }  }    }; |

服务器端的事件更新，见表格：

|  |
| --- |
| *@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"*) |

5.5基于python库xlwt 与xlrd的导入导出Excel的实现

5.5.1 Excel格式定制的分析

5.5.2 导入Excel

导入Excel的处理流程如下所述：

用户点击页面上的import按钮，选择上传要导入的excel文件，通过html 表单发送给app\_route, 相关的import\_excel被调用执行，在import\_excel方法中，获取上传的文件，保存到指定目录下，再调用service模块的Parser类来处理解析excel文件并添加inventory的事项。最后返回到inventory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> |

Js中触发一个bootstrap的modal对话框事件：

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

后台的python实现如下：

|  |
| --- |
| *@app.route*(*'/import-excel'*, methods=[*'POST'*])  def **import\_excel**():  *'''needs to check file type == xlsx etc.*  *if file exists, then server will overrides it by default.*  *'''*  file = request.files[*'choose-excel-file'*]  print file.filename  file\_path = os.path.join(app.config[*'UPLOAD\_FOLDER'*], file.filename)  print file\_path  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'*) |

5.5.3 导出Excel

一个完整的响应流程如下所述：

用户点击页面上的export按钮，js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，service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。

浏览器端代码，如表格所示：

|  |
| --- |
| <a href="/static/export\_file/export.xls" download="export" id="export-excel">  <span class="glyphicon glyphicon-export"></span> Export  </a> |

Js代码，如表格所示：

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

app\_route的代码，如表格所示：

|  |
| --- |
| def **export\_excel**():  invs = InvDao.get\_all\_invs()  data = []  inv\_args = []  for inv in invs:  inv\_args.append(inv.tag)  inv\_args.append(inv.name)  inv\_args.append(inv.PN)  inv\_args.append(inv.SN)  inv\_args.append(inv.shipping)  inv\_args.append(inv.capital)  inv\_args.append(inv.disposition)  inv\_args.append(inv.status)  inv\_args.append(inv.owner)  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) |

其中，Writer.export\_excel(data, file\_path)的实现在前文的service模块已经介绍过，可参见表。

主要参考文献：

[1] 罗海滨,范玉顺,吴澄等.工作流技术综述[J].软件学报,2000,11(7):899-907.

[2] 叶立新,陈闳中,郑航等.基于工作流技术的OA系统模型[J].计算机工程与应用,2000,36(6):90-92,128.DOI:10.3321/j.issn:1002-8331.2000.06.033.

[3] 李伟平,范玉顺.工作流技术在ERP系统中的应用[J].高技术通讯,2004,14(8):56-61.DOI:10.3321/j.issn:1002-0470.2004.08.013.

[4] 葛振坤.ERP研究成果综述与启示[J].金卡工程,2008,12(10):79.

[5] 叶锋.Python最新Web编程框架Flask研究[J].电脑编程技巧与维护,2015,(15):27-28.DOI:10.3969/j.issn.1006-4052.2015.15.011.

[6] 谈华宇,吴昶成,邱小平等.基于Bootstrap框架的动态表单设计与实现[J].无线互联科技,2015,(3):99-100.DOI:10.3969/j.issn.1672-6944.2015.03.041.

[7] “[基于WEB工作流管理系统的设计与实现](http://www.cioage.com/art/200906/79508.htm)”. CIOAge. [2009-06]

[8] Georgakopolous D,Hornick M,Sheth A.An overview of workflow management : from process modelling to workflow automation infrastructure. Distributed and Parallel Databases . 1995

[9] [Office Automation](http://www.inc.com/encyclopedia/office-automation.html). Inc.com. [2013-08-28]

[10] Rouse, Margaret. ["ERP (enterprise resource planning)"](http://searchsap.techtarget.com/definition/ERP). TechTarget. Retrieved July 14, 2015.

[11] Rouse, Margaret "[inventory manangement](http://searchmanufacturingerp.techtarget.com/definition/Inventory-management)". TechTarget. Retrieved July 14, 2015

[12] ["Work Flow Automation"](http://web.archive.org/http:/www.nocsmart.com/index.php?option=com_content&view=article&id=17&Itemid=135). web.archive.org*. Retrieved 31 March 2015*.

[13] Flask框架官方网站. <http://flask.pocoo.org/>

[14] Miguel Grinberg. "[The Flask Mega-Tutorial, Part XI: Email Support](http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xi-email-support)". [2012-12]