

**本科毕业设计（论文）**

题 目 分布式机票动态数据管理系统设计

学院名称 计算机科学与技术学院

专业名称 软件工程

年级班级 1401

学生姓名 林丛丛

指导教师 刘 洁

年 月

摘要

随着人们生活水平的不断提高，每年出国旅行的人数在不断地增加，各个航空公司良莠不齐，数据分散，把航空公司的数据统一获取整理就显得非常有必要了。本系统就是在这种情况下应运而生的。

本系统通过对航空公司官方网站的分析，利用Scrapy+Selenium等多种方式来探索获取对应的航班信息并处理入库，异步高并发的架构提高了数据获取的速率。用Flask+Redis做接口来监控处理爬虫，处理分发任务，以分布式的结构做到保证任务可以不间断，且不受单个爬虫意外的影响。通过Django来做管理后台展示数据，Django自带的人性化的管理后台提高了软件开发效率。用Python+shell做的几个辅助脚本可以做到对爬虫机器的监控更新，服务器CPU内存状态的实时监控通知，可以及时应对被爬取网站的反爬策略和各种变化，提前预知各设备的状态。爬取数据的机子和服务器均采用ubuntu系统，方便统一管理和维护。最终实现了分布式分发任务，爬虫获取任务并爬取数据入库，并在Web端管理页面实现等功能。

关键词：爬虫, 分布式, Django, Scrapy

ABSTRACT

As people living standard unceasing enhancement, each year the number of people traveling abroad is constantly increasing, the airlines, the good and bad are intermingled, scattered data, the airlines have unified access to data it is very necessary.This system emerged in this situation.

This system through the analysis of the airline's official website, a variety of ways such as using Scrapy + Selenium to explore corresponding flight information storage and processing, asynchronous high concurrency architecture to improve the rate of data acquisition. Using Flask+Redis as an interface to monitor and handle crawler processing and distribution tasks, the distributed structure ensures that tasks can be uninterrupted and not affected by individual crawler accidents. The humanized management background of Django improves the efficiency of software development. The shell do several auxiliary in Python scripts can be done to the crawler machine monitoring updates, server CPU memory state real-time monitoring of the notice, can timely response to crawl site of climbing strategy and all sorts of change, to predict in advance the equipment status. Both the machine and the server that crawls data adopt ubuntu system, convenient unified management and maintenance.Finally, the distributed task is realized, the crawler gets the task, crawls the data into the library, and manages the page implementation on the Web side.

Key words: Crawler, Distributed, Django, Scrapy

目录

[1. 前言 1](#_Toc515695944)

[1.1系统研究背景 1](#_Toc515695945)

[1.2国内外研究动态 1](#_Toc515695946)

[1.3 系统概览 1](#_Toc515695947)

[1.4 系统开发的目的和意义 2](#_Toc515695948)

[2.可行性分析 3](#_Toc515695949)

[2.1 管理可行性 3](#_Toc515695950)

[2.2 技术可行性 3](#_Toc515695951)

[2.3 经济可行性 3](#_Toc515695952)

[2.4 本章小结 3](#_Toc515695953)

[3.需求分析 4](#_Toc515695954)

[3.1 总体需求 4](#_Toc515695955)

[3.2 功能需求 4](#_Toc515695956)

[3.3 性能需求 4](#_Toc515695957)

[3.4 开发环境与工具 4](#_Toc515695958)

[3.5 技术介绍 5](#_Toc515695959)

[3.5 本章小结 8](#_Toc515695960)

[4. 系统概要设计 9](#_Toc515695961)

[4.1 系统设计的思想 9](#_Toc515695962)

[4.2系统的功能描述 9](#_Toc515695963)

[4.2.1 爬虫的功能描述 9](#_Toc515695964)

[4.2.2 分布式的功能描述 9](#_Toc515695965)

[4.2.3 后台的功能描述 9](#_Toc515695966)

[4.3 系统的功能设计 10](#_Toc515695967)

[4.3.1 爬虫的功能设计 10](#_Toc515695968)

[4.3.2 分布式的功能设计 11](#_Toc515695969)

[4.3.4 后台的功能设计 11](#_Toc515695970)

[4.4 系统的设计流程 12](#_Toc515695971)

[4.5 数据库设计 13](#_Toc515695972)

[4.5.1 键值对数据 13](#_Toc515695973)

[4.5.2 关系型数据 13](#_Toc515695974)

[4.5.3关系型数据设计的ER图 14](#_Toc515695975)

[4.5.4 系统数据表 15](#_Toc515695976)

[4.5.5 系统数据表建表语句 17](#_Toc515695977)

[4.6 本章小结 18](#_Toc515695978)

[5.系统详细设计 19](#_Toc515695979)

[5.1 爬虫设计 19](#_Toc515695980)

[5.2 分布式设计 21](#_Toc515695981)

[5.3 管理后台设计 21](#_Toc515695982)

[5.4 本章小结 23](#_Toc515695983)

[6 系统实现 24](#_Toc515695984)

[6.1 爬虫的实现 24](#_Toc515695985)

[6.1.1 对任务的获取处理实现 24](#_Toc515695986)

[6.1.2 对数据的分析和处理 25](#_Toc515695987)

[6.1.3 数据入库的实现 27](#_Toc515695988)

[6.2 分布式部分实现 29](#_Toc515695989)

[6.2.1 添加爬虫任务的实现 29](#_Toc515695990)

[6.2.2 暂停爬虫任务的实现 29](#_Toc515695991)

[6.2.3 恢复爬虫任务的实现 30](#_Toc515695992)

[6.2.4 获取爬虫任务的实现 30](#_Toc515695993)

[6.2.5 获取某爬虫对应的任务实现 30](#_Toc515695994)

[6.2.6 添加命令的实现 31](#_Toc515695995)

[6.2.7 获取命令的实现 32](#_Toc515695996)

[6.3 web管理后台的具体实现 32](#_Toc515695997)

[6.3.1 数据模型设计实现 32](#_Toc515695998)

[6.3.2 管理员登录 34](#_Toc515695999)

[6.3.2 管理员管理实现 34](#_Toc515696000)

[6.3.3 管理员对航班信息的管理 35](#_Toc515696001)

[6.4 本章小结 35](#_Toc515696002)

[总结 36](#_Toc515696003)

[致谢 37](#_Toc515696004)

[参考文献 38](#_Toc515696005)

# 1. 前言

## 1.1系统研究背景

生活水平的提高让一个个说走就走的旅行不断地成为现实。然而，远途旅行必备订机票却是一个顶级的技术活，如果没有做好充分的准备，相同的航班差价一两千的比比皆是，关键是有的人家不仅价格比你低

而且订的还是头等舱。而你却只能花更高的价格享受的却是经济舱的服务。

随着互联网的发展，大数据早已渗入到我们生活中的方方面面，如今很多技术的发展和创新都是基于数据，通过对数据进行处理分析，筛选统计，得到各种已知的和推测的信息。数据信息早已成为我们生活和发展中不可分离的重要组成部分。当我们旅行要购买机票时却发现各个航司的飞机的航班信息分散各自的官网对于普通人来说在一个个网站上搜索比价能让你累得要死还分析不好，而去哪儿，携程等OTA平台早已挖好了各种坑等着你往里跳，按照国人的习惯，出次远门不给亲朋好友带点什么都觉得对不起这次旅行。而这个OTA平台大部分票价要比航司官网高，有些会借低点，但是行李，值机，各种坑，有些行李在官网上就是机票里面免费带的，但是OTA平台上却要收百十块钱。所以，一个免费的能即时统计航班信息的系统对我们来说是非常有必要的。

该系统对在数据方面，不断地获取更新数据，并且对于网站的变化能够做到及时地通知处理，保证数据是准确有效的。当用户添加关注时，如果关注价格有变化，可以发邮件通知关注的人。让他及时到对应官网查看买票。极大地满足了懒人的需求。

## 1.2国内外研究动态

1909年11月16日成立的德国飞艇股份公司是世界上第一家航空公司。而我们的国家也是最早从事空运的亚洲国家之一，1929年开始营业的中国航空是由当时的中华民国政府与美国柯蒂斯-莱特公司合资成立。1970年代末， 现代电脑订票系统的出现为航空公司进行成本收益分析带来了巨大的便利，其负作用就是让票价变化更灵活，波动更大。

国外的比较好的比价网站GoogleFlights, KAYAK等，数据很多更新也比较快，但国内访问需要翻墙。国内的航班信息比价网站有Skyscanner，momondo等，在Skyscanner上有很友好的中文界面并且能够直接显示人民币价格，但是订票的时候但是有的是跳转到对应的航班信息官网，有的却是跳转到OTA平台的官网，有的票价要比官网上的高。

## 1.3 系统概览

该系统是在ubuntu上基于Python开发的爬虫实现的有关动态机票信息的系统。Scrapy爬虫实现对数据的获取更新。Flask对redis进行封装实现对任务的分布式管理。Django Web后台实现对数据的友好展示。

## 1.4 系统开发的目的和意义

在物质生活极大得到满足的现实社会，互联网科技的进步让以前许许多多的不可能逐步变为可能，怎么获取让分散的数据清晰明了出集中在用户面前并能根据用户的需求来更新数据是本系统实现的主要功能。该系统相当于是在用户和信息之间搭建了桥梁，既让信息展现在了用户面前，又让用户能够得到并处理信息，当信息有变化时还能及时通知到用户。最大限度地帮助用户解决了买便宜机票难的问题。

# 2.可行性分析

## 2.1 管理可行性

本系统的主要核心部分为爬虫部分，不仅仅数据是动态的，爬取的网站也是动态的，它可能不定时地改变反爬策略，发布新版本等。对此，也有一个完善的脚本来更新处理。Ubuntu系统相对稳定，适合开发，统一的系统可以很方便地操作处理。Web端有固定的环境，固定的版本，所以就不需要耗费太大的精力去维护。

## 2.2 技术可行性

该系统的设计和开发，用的是基于Python语言的不同框架技术以及以网络请求分析的基础上完成的，并且能够在Web界面友好展现， 所以从数据展示， 功能，操作等方面来说都是简单的。该系统的主要目的是能够源源不断地获取并更新数据，除了对数据的更新，还要保证爬虫小机子能够正常运行，不断地获取数据，并能够对爬虫进行更新处理。分布式保证了一个爬虫的异常影响不了任务的执行，数据的获取。对于系统的维护可以部分通过Bash Shell脚本来处理解决，逻辑并不是很复杂；所以技术可行。

## 2.3 经济可行性

本系统的开发工具是PyCharm，用社区版完全免费，现在爬取的数据只是少量数据所以我的一台个人电脑足够。python的所有框架模块均为开源，在PyCharm里可以简单方便地查看源码进行学习和解决，网络上各种关于Python的教程也有许多。所以不需要投入多少的学习成本便可投入开发，所以在经济可行性方面，我也认为是完全可以的。

## 2.4 操作可行性

本系统的开发是主要是由基于Python的爬虫框架Scrapy，Web框架Flask和Django以及前端框架bootstrap一起组合而成的。其中还有一些简单的脚本可以辅助来监控机子状态，控制爬虫，任务等。而对于管理人员来说，只需要在Web端就可以实现对爬虫，任务，数据的管理，监控和各种操作。对于普通用户来说，只需要把自己的关心的数据插入到数据，如果有关于该数据的更新，就自动发送邮件到该用户的绑定邮箱，而用户所需要的操作也是很简单便捷的。所以在操作方面，该系统可行。

## 2.5 法律可行性

该系统是本人自己编写；所用的开发工具是免费开源的；爬取的数据是网站上公开的；数据也没有用于商业用途；使用的技术框架都是开源的，并且可以下载查看源码的，因此不存在侵权行为。本系统web端的用户信息和管理员信息都是通过加密保护的，保证了其安全性，因此不存在泄露隐私的行为。所以该系统在法律上是可行的。

## 2.4 本章小结

本章的主要内容是通过不同的方面来分析该系统的可行性。针对该系统的管理可行性、技术可行性、经济可行性操作可行性和法律可行性进行分析。在管理可行性分析中对该系统的环境，意外处理情况进行分析。在技术可行性分析中对该系统用到的技术安全性、稳定性进行分析。在经济可行性分析中对该系统的学习成本，运行成本，维护成本进行分析。在操作可行性分析中对该系统的管理员和普通用户的操作进行分析。在法律可行性分析中对该系统的技术、开发工具、数据隐私等进行分析。通过各个方面的分析认为该系统是可行的。

# 3.需求分析

## 3.1 总体需求

本系统的是由数据支持的主要系统，要做到对数据的更新，因为要维持数据的相对稳定，所以也要能够做到对爬虫的灵活处理。其次是对数据的展示，搜索等。需求分析就是根据用户的需要完成系统的设计，需求分析主要包括功能需求和性能需求。

## 3.2 功能需求

该系统是基于数据的并对数据进行管理的系统，每爬一次数据就要不断地把数据更新到数据库里面，然后再在web端对数据进行显示。

（1）对任务的管理，能够保证任务队列一直有数据。能够添加任务，暂停任务，删除任务等。

（2）对爬虫的要求，保证爬虫能够不断爬取对方网站并分析处理获取到的响应数据，清洗处理后更新或添加入库

（3）对航班信息的管理，管理员登录后可以删除添加更改航班信息，用户登录后可以搜索航班信息。

（4）对用户的管理， 管理员登录后可以查看、修改、添加、删除用户。

## 3.3 性能需求

（1） 该系统的爬虫部分暂时是对两个航空公司的官方网站进行分析处理，提取入库，为了方便统一管理，均用可联网的ubuntu系统来爬取。

（2） 由于在数据爬取时可能会遇到一些顽固的航司，需要获取通过js加载出来的cookie, 所以需要是桌面版的ubuntu系统用selenium来模拟浏览器获取cookie，所以硬件最好是自带显卡。

## 3.4 开发环境与工具

本系统选择的开发语言为Python2.7, 在Ubuntu下，使用PyCharm集成开发编辑器，通过Charles分析网站及其App，任务存储于Redis数据库，获取到的数据存储于MySQL数据库。爬虫运行系统统一为Ubuntu。

(1) Ubuntu

在开发系统上我选择了最适合Python开发的Ubuntu系统， 可以减少很多Windows上出现的环境、编码等各种很烦琐复杂的问题。减轻了开发的负担。

(2) Pycharm

PyCharm是一个主要用于Python语言开发的集成开发环境，由捷克公司JetBrains开发，它不仅提供代码分析、图形化调试器，集成测试器，还集成版本控制系统：为Mercurial, Git, Subversion, Perforce 和CVS提供统一的用户界面，拥有修改以及合并功能。

(3) Charles

对于网站抓包数据的分析，用浏览器自带的开发者工具足矣，用Charles主要是用来分析vy官网的App端。Charles是跨平台的比较好用的网络封包截取工具。Charles通过将自己设置成系统的网络访问代理服务器，使得所有的网络访问请求都通过它来完成，从而实现对网络请求的截取和分析。

(4) Redis

Redis是支持网络、基于内存、可选持久性的键值对存储数据库，Redis中值的类型不仅限于字符串，还支持字符串列表、无序不重复的字符串集合、有序不重复的字符串集合、键值都为字符串的哈希表。

(5) MySQL

MySQL是一个开放源代码的关系型数据库管理系统，其特性如下：

使用C和C++编写，并使用了多种编译器进行测试，保证了源代码的可移植性。

为多种编程语言提供了API

支持多线程，充分利用CPU资源，支持多用户

优化的SQL查询算法， 有效地提高查询速度。

## 3.5 技术介绍

（1) Python语言

Python是具有解释性、编译性、互动性的面向对象的脚本语言。它具有很强的可读性，比起其他语言经常使用英文关键字，其他语言的一些标点符号，它具有比其它语言更有特色的语法结构，对齐要求相当严格，拥有完善的基础代码库，包含了网络、I/O、GUI、数据库等大量内容，用Python开发， 许多功能可以直接使用。Python还有大量的第三方库。很多网站用Python开发，比如国内的有豆瓣，知乎，麦子学院等，国外的有Google, YouTube,Yahoo! Groups等。

使用Python有如下优点：

简洁

Python的设计哲学就是“优雅”、“明确”，“简单”。语法 简洁清晰，用Python语言能够简单编写算法，并且编写的算法代码非常的优雅，能够让人们容易理解算法实现的原理。Python独特的代码编写风格，采用缩进的方式断句，使得整个代码段很有层次感。

易读写

对其他编程语言很熟悉的程序员或者是有一定编程能力的程序员对Python代码是很容易读懂的，并且很容易上手编程。对Python的代码也可以阅读，并能够理解。

交互性

可以直接从命令行尝试执行我们编写的代码，运行程序，有交互提示。

扩展性

Python自带很多标准库和包，可以通过pip直接下载安装使用。

粘合性

作为一门胶水语言，能够和C, JAVA等主流语言混合使用。

当然，Python并不完美，它也存在一些缺点，运行速度慢，代码不能加密等等。一个程序的瓶颈往往在20%的代码里面，这部分代码可以过CPython，JPython等与其它语言一起使用。所以运行速度慢的问题可以勉强解决。

(2) Scrapy框架

Scrapy是一个以爬取网站数据，提取结构性数据为目的的应用框架。可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。

Scrapy架构

Scrapy是基于事件驱动网络框架Twisted编写的。因此，Scrapy基于并发考虑由非阻塞异步)代码实现，其结构如图３－１所示。

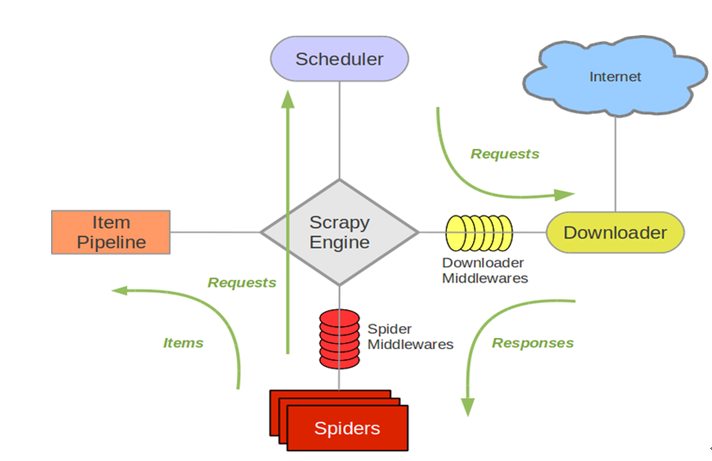


图3－1 Scrapy结构图

②组件：

Scrapy Engine:

引擎负责控制数据流在系统中所有组件中的流动，并且在相应动作发生时触发事件。

调度器(Scheduler):

调度器从引擎接受request并且将它们入请求队列，以便之后 引擎请求它们时可以按序获取队列中的数据。

下载器(Downloader):

下载器负责获取页面数据并提供给引擎， 然后引擎再提供给spider。

Spiders:

Spider是Scrapy用户编写用于分析response并提取item(即获取到的item)或额外跟进的URL的类。每个spider负责处理一个特定的网站。

Item Pipeline:

Item Pipeline 负责处理被spider提取出来的item.

中间件:

中间件包括下载器中间件(Downloader middlewares)和Spider中间件(Spider middlewares)。下载器中间件是在引擎和下载器之间的特定钩子(specific hook), 处理Downloader传递给引擎的response；Spider中间件是在引擎及Spider中间的特定钩子， 处理spider的输入(response)和输出(items及requests)。 两者都提供了一个简便的机制，通过插入自定义代码来拓展Scrapy功能。

(3) Flask框架

Flask是一个用python编写的轻量级Web应用框架。基于Werkzeug WSGI工具箱和Jinja2模板引擎。

Flask的一个显著特点就是“微”， “微”意味着Flask旨在保持核心的简单，同时又易于扩展。默认情况下，Flask不包含数据库抽象层、表单验证，或是其它已有库可以做到的功能。然而Flask保留了扩增的弹性，可以用Flask-extension加入这些功能这也是为什么我要用Flask做API接口来处理任务以及爬虫等。

当然，我也用了Flask的一个拓展flask\_apscheduler来设置循环任务。

(4) 分布式爬虫原理

通过Flask对redis任务队列进行封装，Srapy爬虫不断地向redis请求任务并添加到调度器里，然后爬虫进行数据爬取，所有爬虫各自独立互不影响。

(5) Django框架

Django是一个开放源代码的Web应用框架，由Python写成。采用MVT的软件设计模式，就是模型Model, 视图View和模板Template。诞生于新闻网站的环境中，因此它提供了很多的特性， 非常适合内容类的网站。Djanog尤其擅长于动态内容管理系统。它的主要目标是使得开发复杂的、数据库驱动的网站变得简单。Django尤其注重组件的重用性和“可插拔性”，敏捷开发和DRY法则。

MTV各部分之间的关系：

模型为数据处理层，或者数据模块

使用ORM框架， 一个Model对映数据库的一张表，包含一些基本的字段以及数据的一些行为，属性等。

视图为执行代码所在的模块，是代码主要处理逻辑的地方。

每一个响应对应一个函数，每一个响应也都对应一个URL.

模板为表现层

Django的模板语言相当强大，可以轻松地将设计, 逻辑和内容分隔开。还可以实现模板的包含和继承, 并且是可拓展的。

(6) Bootstrap框架

Bootstrap是基于HTML, CSS, JAVASCRIPT的一个用于快速开发Web应用程序和网站的前端框架。它的优点在于将常用的CSS组件还有JavaScript插件进行了完整的封装，不管你有没有前端开发经验，都能让你轻松地掌握使用，从而提高开发效率。其特点如下：

抽象出常用的CSS样式，高再可用性，高移植性。

有丰富的组件，Bootstrap有大量可复用的组件这些组件能够帮助我们快速地完成一个网站的前端设计，实现一个漂亮的个性化网站。这些组件包括字体、图标、导航、弹出框、下拉菜单等多个功能。

大量的JavaScript插件，比如过渡效果、滚动监听、标签页、Collaps和carousel等。可以单次引入这些插件，也可以一次全部引入。

高兼容性，可以兼容市面上流行的大部分浏览器以及移动设备。

## 3.5 本章小结

本章主要介绍了该系统的需求分析，包括功能需求，性能需求，开发运行环境以及用到的技术。功能上主要是对数据，对爬虫，对任务的一些处理操作。由于ubuntu系统适合开发，尤其是Python开发，且稳定易维护，所以ubuntu系统是我的爬虫运行环境的不二选择。通过对该系统的技术的研究与学习， 让我对网络请求，web开发以及分布式的结构原理都有了深刻的了解。

# 4. 系统概要设计

## 4.1 系统设计的思想

本系统主要包括爬虫，分布式和后台三个大部分。爬虫是基础，是重中之重，为了保证爬虫的正常运行，需要对爬虫的编写辅助脚本监控CPU和获取命令并执行。分布式主要是对redis库里的任务队列进行包装，方便爬虫获取并及时给队列添加任务。后台主要进行对数据的友好展示。

## 4.2系统的功能描述

### 4.2.1 爬虫的功能描述

（1）从任务队列获取任务

（2）添加cookies, 头信息，UserAgent来伪装自己

（3）通过爬取指定目标网站来获取需要的数据

（4）对数据进行处理，格式化

（5）判断库里是否有数据，有的话就更新，没有就插入数据库

（6）定时心跳，证明自己是活着的，同时获取控制命令并执行

### 4.2.2 分布式的功能描述

任务爬虫管理一个航司的所有任务，包括对应爬虫的所有小任务

（1）添加爬虫任务

（2）暂停爬虫任务

（3）恢复爬虫任务

（4）获取所有爬虫任务

（5）获取某爬虫对应的部分小任务

（6）给爬虫主机添加命令

（7）获取某主机对应的所有命令

### 4.2.3 后台的功能描述

(1)管理员登录到系统后台才能进行操作，可进行的操作有：

查看航班信息

修改航班信息

删除某个航班信息

查看用户信息

删除用户

更改用户信息

查看特别关注

修改特别关注

删除特别关注

(2)用户登录到系统才能进行操作，可进行的操作有：

查看航班信息

搜索航班信息

添加到特别关注

删除特别关注

## 4.3 系统的功能设计

### 4.3.1 爬虫的功能设计

爬虫的功能模块图如图4－1所示，首先爬虫获取任务，其次伪装自己，避免被反爬，然后再获取数据，对数据处理然后到Pipeline里入库处理。额外的定时心跳和获取命令并执行只是为了方便对爬虫的监控管理。

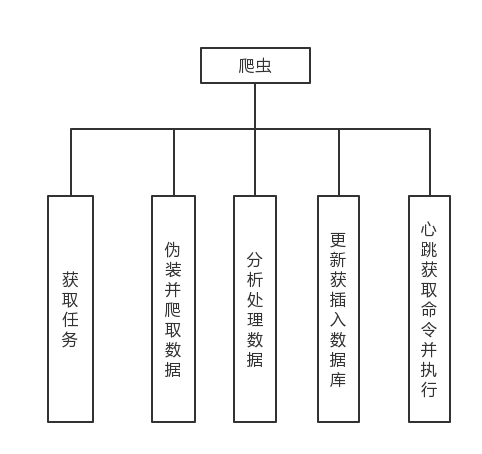


图4－1 爬虫功能模块图

### 4.3.2 分布式的功能设计

如图4－2所示，主要包括添加爬虫任务，暂停爬虫任务，恢复爬虫任务，获取所有爬虫任务，获取某爬虫对应的部分小任务，添加命令，获取某主机对应的所有命令七部分。

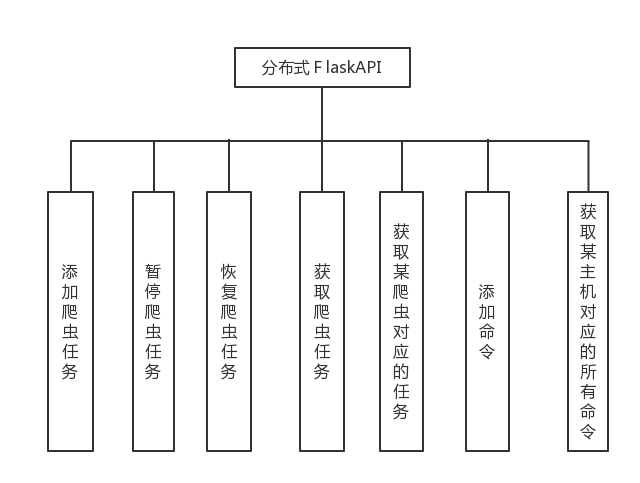


图4－2 分布式接⼝模块图

### 4.3.4 后台的功能设计

后台主要有管理员和普通用户组成，管理员有对所有数据更改信息的权力，用户主要有对航班信息进行查找对自己的特别关注进行操作的权限。如图4－3所示。

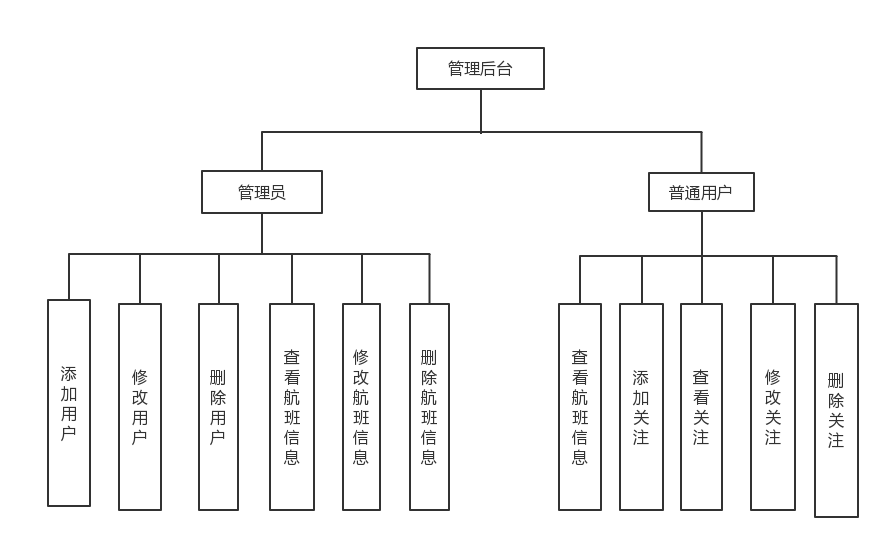


图4－3 后台功能模块图

## 4.4 系统的设计流程

整个系统主要是围绕着数据来设计，最终实现对数据的获取和友好展示，系统的整体数据流程设计如图4－4所示。

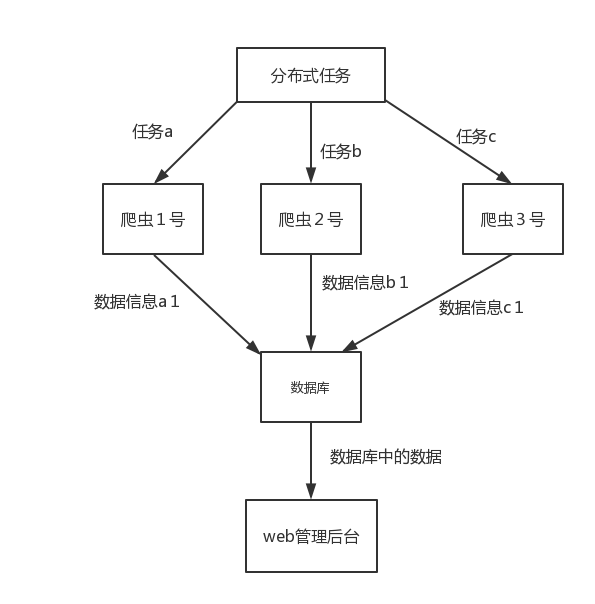


图4－4 系统整体数据流程图

## 4.5 数据库设计

### 4.5.1 键值对数据

本系统的任务，状态，以及命令均用的redis数据库，为键值对类型，键为字符串，值有多种形式，可以包括字符串、列表、散列、集合、有序集合、HyperLogLog。任务和命令用的是字符串和列表的组合，左出右进实现队列的结构。爬虫状态用的是字符串和字符串的组合。

### 4.5.2 关系型数据

为了提高开发效率，用户和管理员用的是Django自带的认证系统。自己额外建的表有航班信息表ticket\_tickets, 关注表ticket\_concerned。数据关系模型如下：

航班信息表（航班号，起飞时间，到达时间，出发机场，到达机场，货币种类，总票价，税价，净票价，剩余座位数，舱位，航空公司，是否中转， 添加时间，更新时间，以及航段信息）

航段信息为json格式的字符串（航班号，机型，该航段在该航线中的次序，出发时间，到达时间，航空公司，出发机场，到达机场，该航段的剩余座位数，该航段持续时长，起始航站楼）

关注表（出发机场，到达机场，用户，起始时间，结束时间）

Django自带用户表（用户名， first\_name, last\_name，邮箱，密码，是否可以访问管理界面，是否活跃，是否有全部权限，上一次的登录时间，用户创建的时间）

还有其它的一些Django自带的权限表，日志表等。

### 4.5.3关系型数据设计的ER图

概念设计也是数据库设计中尤其重要的一部分，即画出E－R图，这样不管你对该系统熟悉与否，都能保证你能看懂该系统的数据结构，数据分析。该系统的ER图如下所示。

（1）航班信息ER图如下图4－5所示。

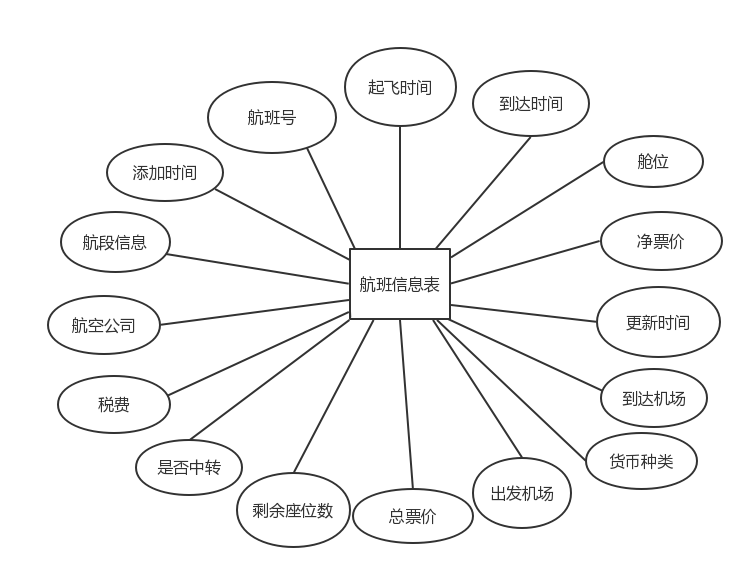


图4－5 航班信息ER图

（2）关心信息表ER图如图4－6所示。

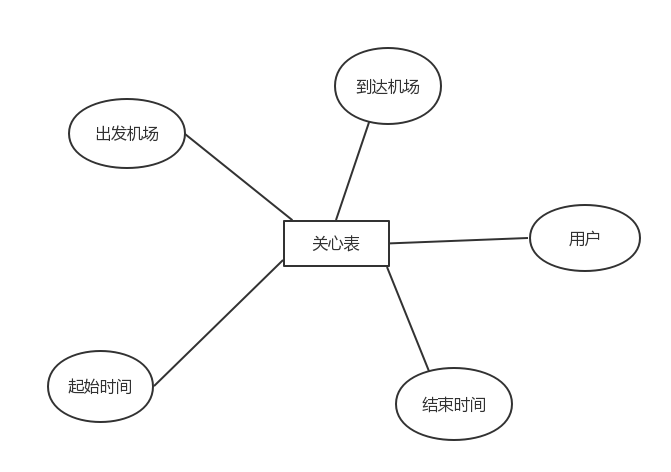


图4－6 关心表ER图

（3）用户信息表ER图如图4－7所示。

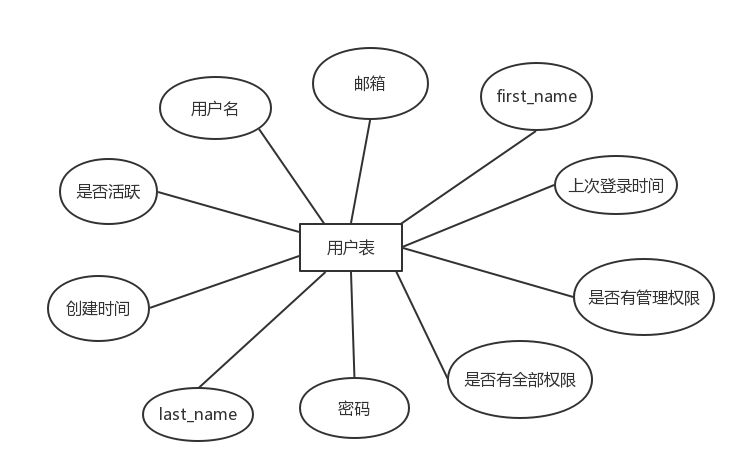


图4－7 用户信息表ER图

### 4.5.4 系统数据表

系统数据表主要说明了给系统中比较重要的表的建设，主要说明了每张表有哪些字段，每个字段的类型及其对应的长度，是不是主键等。

1. 航班信息表：表名 “ticket\_tickets”,表结构如表4－8所示。

表4－8 航班信息表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 数据类型 | 字段长度与格式 | 关键字 |
| id | int | 11 | 是 |
| flightNumber | varchar | 20 | 否 |
| depTime | int | 11 | 否 |
| arrTime | int | 11 | 否 |
| depAirport | varchar | 6 | 否 |
| arrAirport | varchar | 6 | 否 |
| currency | varchar | 6 | 否 |
| adultPrice | int | 11 | 否 |
| adultTax | int | 11 | 否 |
| netFare | int | 11 | 否 |
| maxSeats | int | 11 | 否 |
| cabin | varchar | 10 | 否 |
| carrier | varchar | 5 | 否 |
| isChange | int | 11 | 否 |
| getTime | int | 11 | 否 |
| addTime | int | 11 | 否 |
| segments | longtext | 4Gb | 否 |
| site | varchar | 256 | 否 |

1. 关心信息表：表名“ticket\_concerned”,表结构如表4－9所示。

表4－9 关心信息表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 数据类型 | 字段长度与格式 | 关键字 |
| id | int | 11 | 是 |
| depAirport | varchar | 10 | 否 |
| arrAirport | varchar | 10 | 否 |
| startTime | datetime | 6 | 否 |
| endTime | datetime | 6 | 否 |
| user\_id | int | 11 | 否 |

1. 用户表：表名“auth\_user”,表结构如表4－10

表4－10 用户表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 数据类型 | 字段长度与格式 | 关键字 |
| id | int | 11 | 是 |
| password | varchar | 128 | 否 |
| last\_login | datetime | 6 | 否 |
| is\_superuser | tinyint | 1 | 否 |
| username | varchar | 150 | 否 |
| first\_name | varchar | 30 | 否 |
| last\_name | varchar | 30 | 否 |
| email | varchar | 254 | 否 |
| is\_staff | tinyint | 1 | 否 |
| is\_active | tinyint | 1 | 否 |
| date\_joined | datetine | 6 | 否 |

### 4.5.5 系统数据表建表语句

第一个表，也是最重要的表即航班信息表，主要存储航班信息。第二个表是关心表，即某用户所关心的数据。第三个表是用户表，Django认证系统自带的用户表。用于认证用户，存储用户信息等。

1. 航班信息表建表语句：

CREATE TABLE `ticket\_tickets` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`flightNumber` varchar(20) NOT NULL,

`depTime` int(11) NOT NULL,

`arrTime` int(11) NOT NULL,

`depAirport` varchar(6) NOT NULL,

`arrAirport` varchar(6) NOT NULL,

`currency` varchar(6) NOT NULL,

`adultPrice` int(11) NOT NULL,

`adultTax` int(11) NOT NULL,

`netFare` int(11) NOT NULL,

`maxSeats` int(11) NOT NULL,

`cabin` varchar(10) NOT NULL,

`carrier` varchar(5) NOT NULL,

`isChange` int(11) NOT NULL,

`getTime` int(11) NOT NULL,

`addTime` int(11) NOT NULL,

`segments` longtext NOT NULL,

`site` varchar(256) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1164 DEFAULT CHARSET=utf8

1. 关心信息数据表建表语句:

CREATE TABLE `ticket\_concerned` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`depAirport` varchar(10) NOT NULL,

`arrAirport` varchar(10) NOT NULL,

`startTime` datetime(6) NOT NULL,

`endTime` datetime(6) NOT NULL,

`user\_id` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `ticket\_concerned\_user\_id\_cb659c5f\_fk\_auth\_user\_id` (`user\_id`),

CONSTRAINT `ticket\_concerned\_user\_id\_cb659c5f\_fk\_auth\_user\_id` FOREIGN KEY (`user\_id`) REFERENCES `auth\_user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

1. 用户信息表

CREATE TABLE `ticket\_concerned` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`depAirport` varchar(10) NOT NULL,

`arrAirport` varchar(10) NOT NULL,

`startTime` datetime(6) NOT NULL,

`endTime` datetime(6) NOT NULL,

`user\_id` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `ticket\_concerned\_user\_id\_cb659c5f\_fk\_auth\_user\_id` (`user\_id`),

CONSTRAINT `ticket\_concerned\_user\_id\_cb659c5f\_fk\_auth\_user\_id` FOREIGN KEY (`user\_id`) REFERENCES `auth\_user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

## 4.6 本章小结

本章主要是对该系统的概要设计，包括功能概述，功能设计，整体流程设计，数据库设计等。通过系统的功能设计可以看出该系统各部分的大致功能，通过对该系统的整体流程设计可以了解该系统的整体功能实现，整体架构。

# 5.系统详细设计

## 5.1 爬虫设计

爬虫要做到获取任务，并根据任务获取到对应的数据，并对数据进行分析，格式化处理，然后根据出发时间和航班号该数据是否在库内，如果在就更新数据，不在就插入数据。此过程为多线程异步处理。

爬虫流程图如图5-1所示：

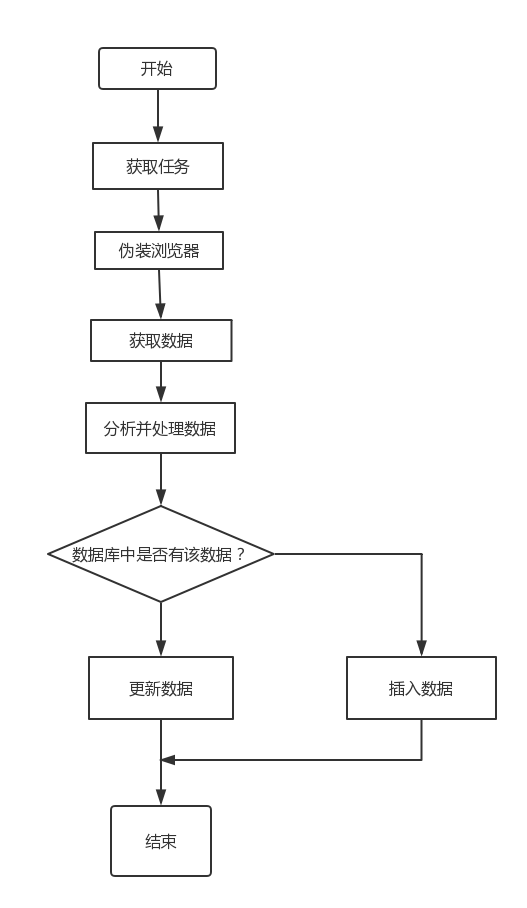


图5－1 爬虫获取数据流程图

## 5.2 分布式设计

本系统的分布式主要是对爬虫任务的不断分发。其主要流程设计如图5－2所示：

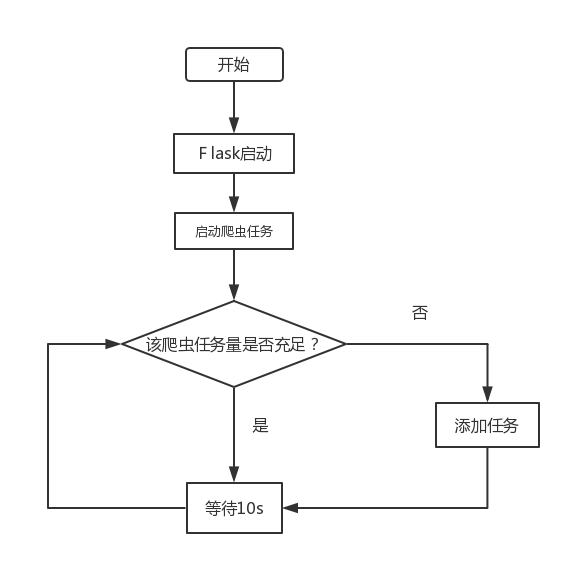


图5－2 分布式接口任务分发流程图

## 5.3 管理后台设计

该后台主要实现对数据的友好展示，并能够方便管理员对数据的管理。Django自带的登录界面，很简洁，只需填入用户名和密码即可。如图5－3所示。

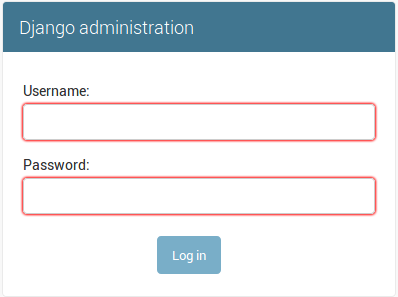


图5－3 登录界面

登录进去之后，也是Django后台自带的界面，简洁大方。包括两人个模块，第一个是认证模块，实现对用户权限的管理。第二个是ticket模块，实现对数据的管理。如图5－4所示。

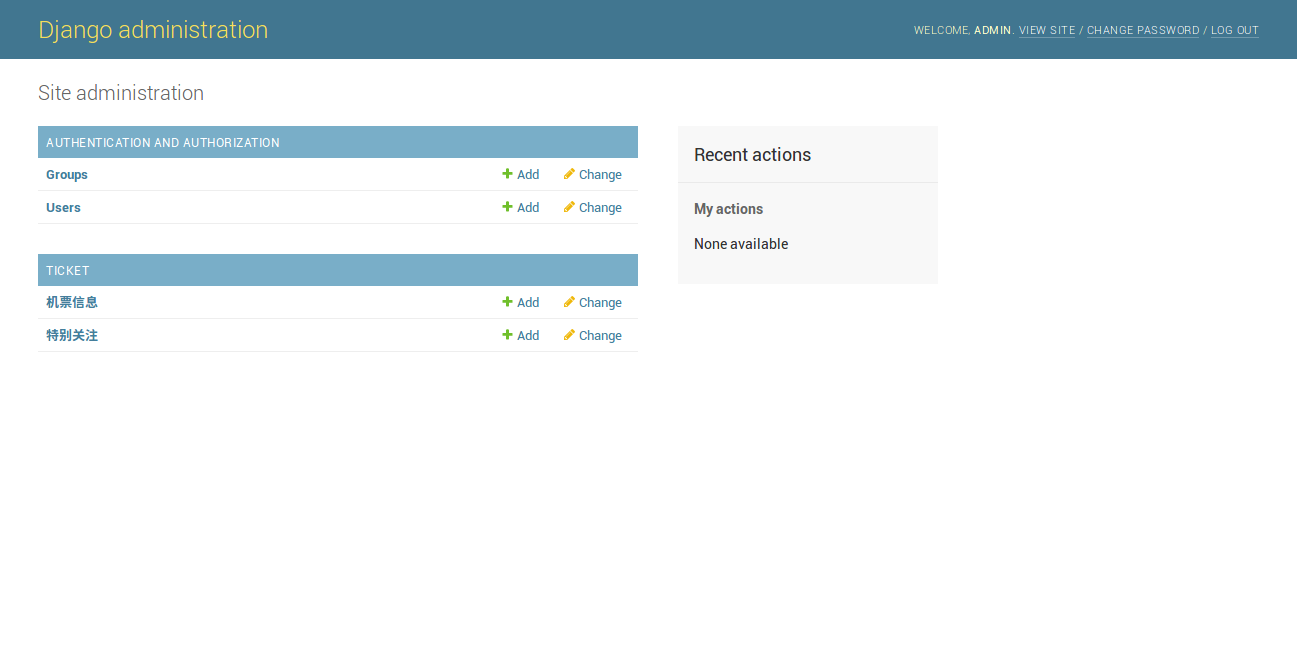


图５－４数据管理界面

管理员可以修改机票信息，如图5－5所示。

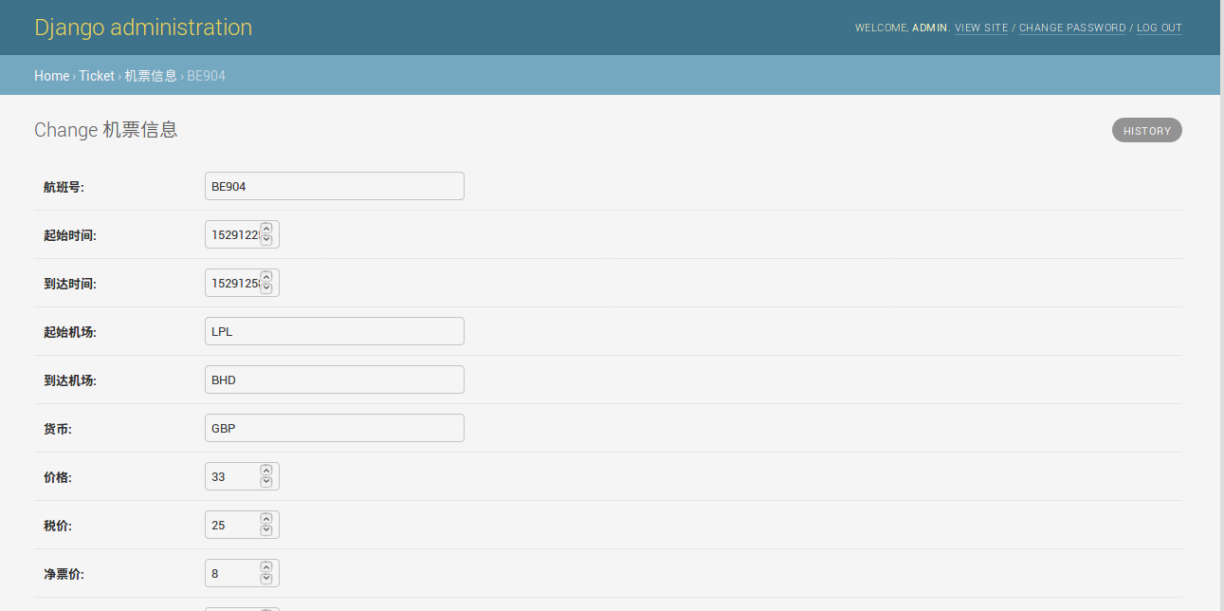


图5-5 修改数据界面

也可以修改用户信息。如图5－6所示。

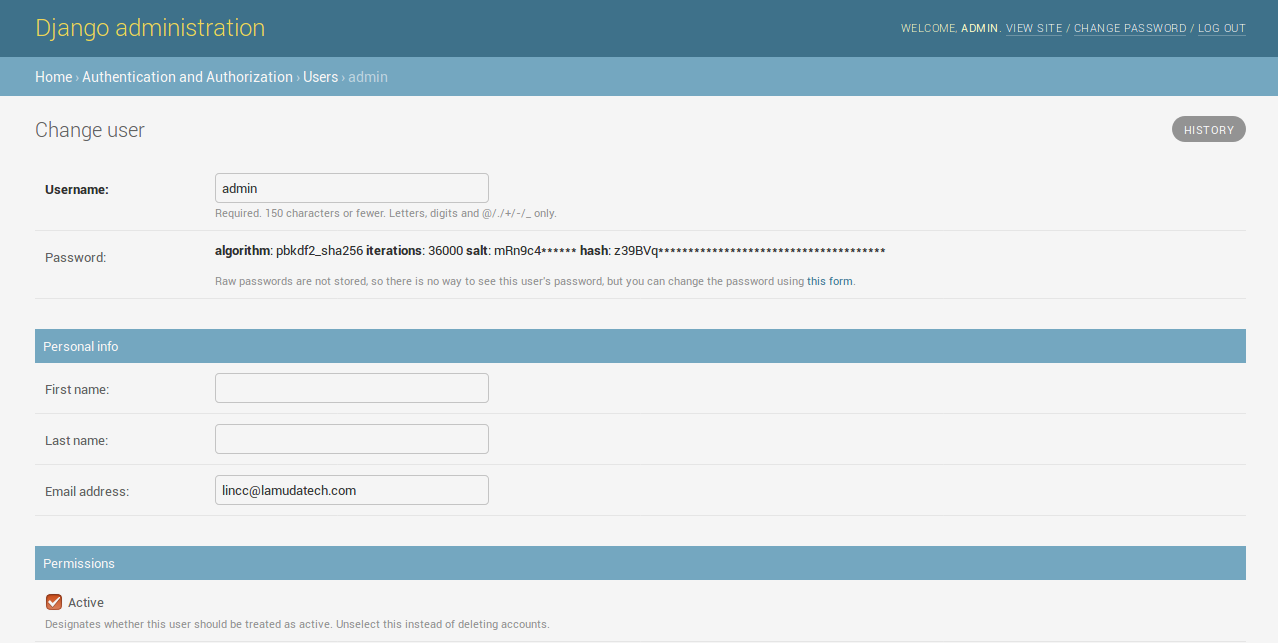


图5－6 修改用户信息界面

## 5.4 本章小结

本章对该系统进行了详细地分析与设计，其主要包含爬虫部分设计，分布式部分设计以及后台管理部分。通过对爬虫部分的流程设计可以详细地了解爬虫爬取数据的运行流程，通过对分布式的详细分析，可以了解分布式原理和redis库的处理，也充分地了解了爬虫，分布式，管理后台之间的交互过程。

# 6 系统实现

## 6.1 爬虫的实现

爬虫要实现对任务的获取，并能不断地抓取稳定的数据,一分钟一百多条，速度不算快，但是如果稳定的话也还算可以。用框架最大的好处是不用自己造轮子，只需关注功能部分即可。每个航司的具体实现也不太一样，这里以BE为例，如图6－1所示。

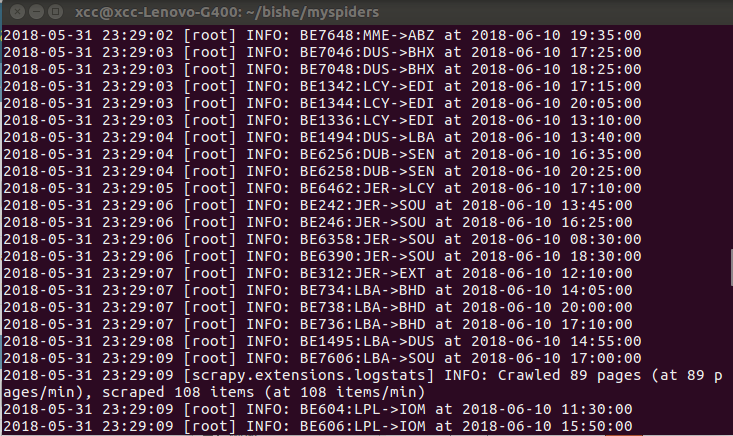


图6－1 数据爬取情况图

### 6.1.1 对任务的获取处理实现

该部分主要实现对爬虫任务的获取，并对获取到的任务数据进行处理，判断是否有效（即日期是否是过去的日期），如果没有获取到数据，则休息10s后重新获取。如果日期无效则忽略，重新获取，遍历获取到的所有任务并生成对应的url，并发送请求。代码实现如下：

def start\_requests(self):  
 while True:  
 result = pubUtil.getUrl('BE', 10)  
 if not result:  
 logging.info('no task! sleep 10s...')  
 time.sleep(10)  
 continue  
 for data in result:  
 (dt, dep, to) = pubUtil.analysisData(data) # 把获取到的data格式化  
 if pubUtil.dateIsInvalid(dt):  
 continue  
 temp = {  
 'depart': dep,  
 'arr': to,  
 'departing': dt,  
 'returning': '',  
 'promo-code': '',  
 'adults': 1,  
 'teens': 0,  
 'children': 0,  
 'infants': 0  
 }  
 try:  
 params = urllib.parse.urlencode(temp)  
 except:  
 params = urllib.urlencode(temp)  
  
 url = '%s%s/%s?%s' % (self.start\_urls, dep, to, params)  
 yield scrapy.Request(url, callback=self.parse, dont\_filter=True,  
 errback=lambda x: self.download\_errback(x, url))

### 6.1.2 对数据的分析和处理

每一次请求获取到的数据都冗杂且有许多多余的，这部分就需要对数据进行筛选，分析，处理，格式化。具体实现代码如下：

def parse(self, response):  
 try:  
 content = json.loads(response.text)  
 except:  
 self.isOK = False  
 logging.info('change IP....')  
 return  
 if 'inbound' in content.keys():  
 return  
 self.isOK = True  
 li = content['outbound']  
 for i in range(len(li)):  
 outbound = li[i]  
 item = MySpidersItem()  
 if outbound['totalHighestAdultGrossFare'] == 0: # 该航班机票已售空  
 continue  
 if 'low' not in outbound.keys(): # 依次查找此时的最低票价  
 if 'medium' not in outbound.keys() or outbound['medium']['flights'][0]['isEligible'] is False:  
 low = outbound['high']  
 else:  
 low = outbound['medium']  
 else:  
 low = outbound['low']  
  
 flights = low['flights']  
 departTime = outbound['departDate'][:10] + ' ' + outbound['departureTime'] + ':00'  
 ti = time.mktime(time.strptime(departTime, '%Y-%m-%d %H:%M:%S'))  
 row = {}  
 row['depTime'] = ti  
 destTime = outbound['arriveDate'][:10] + ' ' + outbound['destinationTime'] + ':00'  
 ti = time.mktime(time.strptime(destTime, '%Y-%m-%d %H:%M:%S'))  
 row['arrTime'] = ti  
 row['depAirport'] = outbound['depart']  
 row['arrAirport'] = outbound['dest']  
 row['currency'] = outbound['currency']  
 row['adultPrice'] = float(low['totalAdultGrossFare'])  
 row['adultTax'] = float(low['totalAdultTaxes'])  
 row['netFare'] = float(low['totalAdultNetFare'])  
 num = outbound['flightCount']  
 segments = []  
 if num > 1: # 判断是否有中转  
 continue  
 else:  
 segments.append(self.analysisSegment(flights[0].copy()))  
 row['maxSeats'] = flights[0]['seatsAvailable']  
 row['cabin'] = flights[0]['fare']['fareClass']  
 row['carrier'] = flights[0]['flightNumber'][:2]  
 row['flightNumber'] = flights[0]['flightNumber']  
 row['isChange'] = 1  
 row['segments'] = json.dumps(segments)  
 row['getTime'] = time.time()  
 item.update(row)  
 yield item

### 6.1.3 数据入库的实现

获取到的数据最终要存储于MySQL数据库，Scrapy里的Pipeline是对获取并处理过的数据的最终处理。在本项目中，该类要实现连接数据库，判断数据库中是否有同日同次航班信息，如果有，就更新数据库，如果没有就插入该信息到数据库。具体实现代码如下：

class MyspidersPipeline(object):  
 def \_\_init\_\_(self):  
 self.db = self.\_get\_con()  
 self.cursor = self.db.cursor()  
 def update\_item(self,table\_name, row):  
 sql = "SELECT id FROM %s WHERE flightNumber='%s' AND depTime=%s" % (table\_name, row['flightNumber'], row['depTime'])  
 self.cursor.execute(sql)  
 result = self.cursor.fetchone()  
 if result:  
 row\_id = result[0]  
 sql\_update = "UPDATE %s SET %s " % (table\_name, ', '.join('{}=%s'.format(k) for k in row))  
 sql\_update += "WHERE id = %s" % row\_id  
 values = tuple(row[key] for key in row)  
 try:  
 self.cursor.execute(sql\_update, values)  
 except:  
 logging.error('update item error')  
 return True  
 return False  
 def addRow(self, tablename, rowdict):  
 rowdict['addTime'] = rowdict.get('getTime')  
 keys = rowdict.keys()  
 columns = ", ".join(keys)  
 values\_template = ", ".join(["%s"] \* len(keys))  
 sql = "insert into %s (%s) values (%s)" % (tablename, columns, values\_template)  
 values = tuple(rowdict[key] for key in keys)  
 try:  
 self.cursor.execute(sql, values)  
 logging.info('%s:%s->%s at %s' % (rowdict.get('flightNumber'), rowdict.get('depAirport'), rowdict.get('arrAirport'), time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(rowdict.get('depTime')))))  
 except:  
 traceback.print\_exc()  
 logging.error('add error')  
 def process\_item(self, item, spider):  
 try:  
 self.db.ping()  
 except:  
 self.db = self.\_get\_con()  
 self.cursor = self.db.cursor()  
 if not self.update\_item('ticket\_tickets', item):  
 self.addRow('ticket\_tickets', item)  
  
 @staticmethod  
 def \_get\_con():  
 db = MySQLdb.connect(host=settings.DB\_HOST, user=settings.DB\_USER, passwd=settings.DB\_PWD,  
 port=settings.DB\_PORT, db=settings.DB\_NAME)  
 db.autocommit(True)  
 return db

## 6.2 分布式部分实现

分布式部分主要是Flask对Redis的封装，然后通过接口与爬虫和管理后台进行交互，其主要原因是redis库没有用户密码，只需通过ip地址和端口就能进行访问。这样做有很大的安全隐患，很容易就被一些黑客扫描到。所以通过接⼝封装，redis库不对外开放，只允许本地访问，保证了数据库服务器的安全性。

### 6.2.1 添加爬虫任务的实现

添加爬虫任务主要是添加对应航司的任务，添加的时候需要提前判断一下，如果该任务已经添加，就不再添加，具体代码实现如下

@app.route('/addjob', methods=['GET', 'POST'])  
def addjob():  
 data = {}  
 carrier = request.args.get('carrier').lower()  
 if not carrier:  
 data['status'] = 1  
 data['msg'] = 'no carrier, add job error!!!'  
 else:  
 jobs = scheduler.get\_jobs()  
 for job in jobs:  
 if job.id == carrier:  
 break  
 else:  
 scheduler.add\_job(func=pushtask, id=carrier, trigger='interval', seconds=10, args=[carrier])  
 data['status'] = 0  
 data['msg'] = 'carrier %s has added !' % carrier  
 data.setdefault('status', 2)  
 data.setdefault('msg', "carrier '%s' had existed !!!" % carrier)  
 return json.dumps(data)

### 6.2.2 暂停爬虫任务的实现

暂停爬虫任务主要是通过APScheduler的内置方法pause\_job来实现，其具体实现代码如下：

@app.route('/pause')  
def pausejob():  
 ret = {}  
 carrier = request.args.get('carrier').lower()  
 scheduler.pause\_job(carrier)  
 ret['status'] = 0  
 ret['msg'] = 'pause success!'  
 return json.dumps(ret)

### 6.2.3 恢复爬虫任务的实现

恢复爬虫任务的实现主要是通过APScheduler的内置方法resume\_job,其具体实现代码如下：

@app.route('/resume')  
def resumejob():  
 ret = {}  
 carrier = request.args.get('carrier').lower()  
 scheduler.resume\_job(carrier)  
 ret['status'] = 0  
 ret['msg'] = 'resume success!'  
 return json.dumps(ret)

### 6.2.4 获取爬虫任务的实现

获取爬虫任务可以看到此时正在运行的任务爬虫有哪几个，主要提供给管理后台来管理。具体实现代码如下：

@app.route('/getjobs')  
def getstatus():  
 ret = {}  
 jobs = scheduler.get\_jobs()  
 ret['status'] = 0  
 ret['jobs'] = jobs  
 return json.dumps(ret)

### 6.2.5 获取某爬虫对应的任务实现

这个接口主要提供给Spiders， 可以做到按要求按量分发任务。其具体代码实现如下：

@app.route('/gettask')  
def gettask():  
 ret = {}  
 carrier = request.args.get('carrier').lower()  
 num = int(request.args.get('num', 1))  
 if not db.llen(carrier):  
 ret['status'] = 1  
 ret['msg'] = 'this carrier is not exist'  
 ret['data'] = []  
 else:  
 data = []  
 min\_num = db.llen(carrier) if db.llen(carrier) < num else num  
 for i in range(0, min\_num):  
 data.append(db.lpop(carrier))  
 ret['status'] = 0  
 ret['data'] = data  
 return json.dumps(ret)

### 6.2.6 添加命令的实现

这个接口主要用于对爬虫的控制和管理，请求接口并携带命令，服务器接收到命令就执行入库操作。其具体代码实现如下：

@app.route('/pushcmd', method=['GET', 'POST'])  
def pushcmd():  
 host = request.args.get('carrier').lower()  
 datas = request.form  
 ret = {}  
 if not datas or not len(datas):  
 ret['status'] = 2  
 ret['msg'] = 'data is empty !'  
 elif len(datas) > 100:  
 ret['status'] = 1  
 ret['msg'] = 'data is too much!'  
 else:  
 item = 0  
 for data in datas:  
 cmds = data.get('cmds')  
 devices = data.get('devices')  
 if not cmds or not devices or not len(cmds) or not len(devices):  
 continue  
 for cmd in cmds:  
 for device in devices:  
 item += 1  
 db.rpush(device, cmd)  
 ret['status'] = 0  
 ret['msg'] = 'has pushed %s items' % item  
 return json.dumps(ret)

### 6.2.7 获取命令的实现

这个功能主要用于对爬虫的维护和管理，提供给对应的爬虫机子，让爬虫机子执行对应的命令。其具体实现代码如下

@app.route('/getcmd')  
def getcmd():  
 ret = {}  
 host = request.args.get('host').lower()  
 cmd = []  
 while db.llen(host):  
 cmd.append(db.lpop())  
 if not len(cmd):  
 ret['status'] = 1  
 ret['msg'] = 'there is no command!'  
 ret['data'] = []  
 else:  
 ret['status'] = 0  
 ret['msg'] = ''  
 ret['data'] = cmd  
 return json.dumps(ret)

## 6.3 web管理后台的具体实现

对于管理后台，主要实现管理员的登录，并对数据进行管理。禀着Django不自己造轮子的原则，就借用的Django自带的管理后台，即Admin模块，具体实现代码均在django.contrib下， 数据模型自己定义。

### 6.3.1 数据模型设计实现

数据模型主要包括机票信息和关注信息，用户表则用Django自带的认证系统，其具体设计实现代码如下。

class Tickets(models.Model):  
 flightNumber = models.CharField('航班号', max\_length=20)  
 depTime = models.IntegerField('起始时间')  
 arrTime = models.IntegerField('到达时间')  
 depAirport = models.CharField('起始机场', max\_length=6)  
 arrAirport = models.CharField('到达机场', max\_length=6)  
 currency = models.CharField('货币', max\_length=6)  
 adultPrice = models.IntegerField('价格', default=0)  
 adultTax = models.IntegerField('税价', default=0)  
 netFare = models.IntegerField('净票价', default=0)  
 maxSeats = models.IntegerField("座位数", default=0)  
 cabin = models.CharField('舱位', max\_length=10)  
 carrier = models.CharField('航司', max\_length=5)  
 isChange = models.IntegerField('是否中转')  
 getTime = models.IntegerField('更新时间')  
 addTime = models.IntegerField('添加时间')  
 # fromCity = models.CharField('出发城市', max\_length=10)  
 # toCity = models.CharField('到达城市', max\_length=10)  
 segments = models.TextField('航班信息')  
 site = models.CharField('站点', null=True, max\_length=256)  
  
 def \_\_str\_\_(self):  
 return self.flightNumber  
  
 class Meta:  
 verbose\_name = '机票信息'  
 verbose\_name\_plural = '机票信息'  
 ordering = ['getTime']  
  
class Concerned(models.Model):  
 depAirport = models.CharField('出发机场', max\_length=10)  
 arrAirport = models.CharField('到达机场', max\_length=10)  
 user = models.ForeignKey('auth.User', blank=True, null=True, verbose\_name='用户')  
 startTime = models.DateTimeField('起始时间', auto\_now=True)  
 endTime = models.DateTimeField('结束时间', auto\_now=True)  
  
 def \_\_str\_\_(self):  
 return self.user  
  
 class Meta:  
 verbose\_name = '特别关注'  
 verbose\_name\_plural = '特别关注'

### 6.3.2 管理员登录

管理员登录只需填写用户名和密码即可实现效果如图6－2所示。

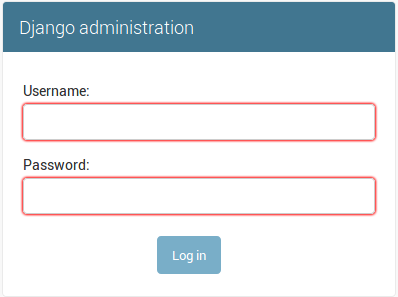


图6－2 登录界面

### 6.3.2 管理员管理实现

管理员主要实现对用户的管理和对航班机票信息的管理，具体实现效果如图6－3所示。

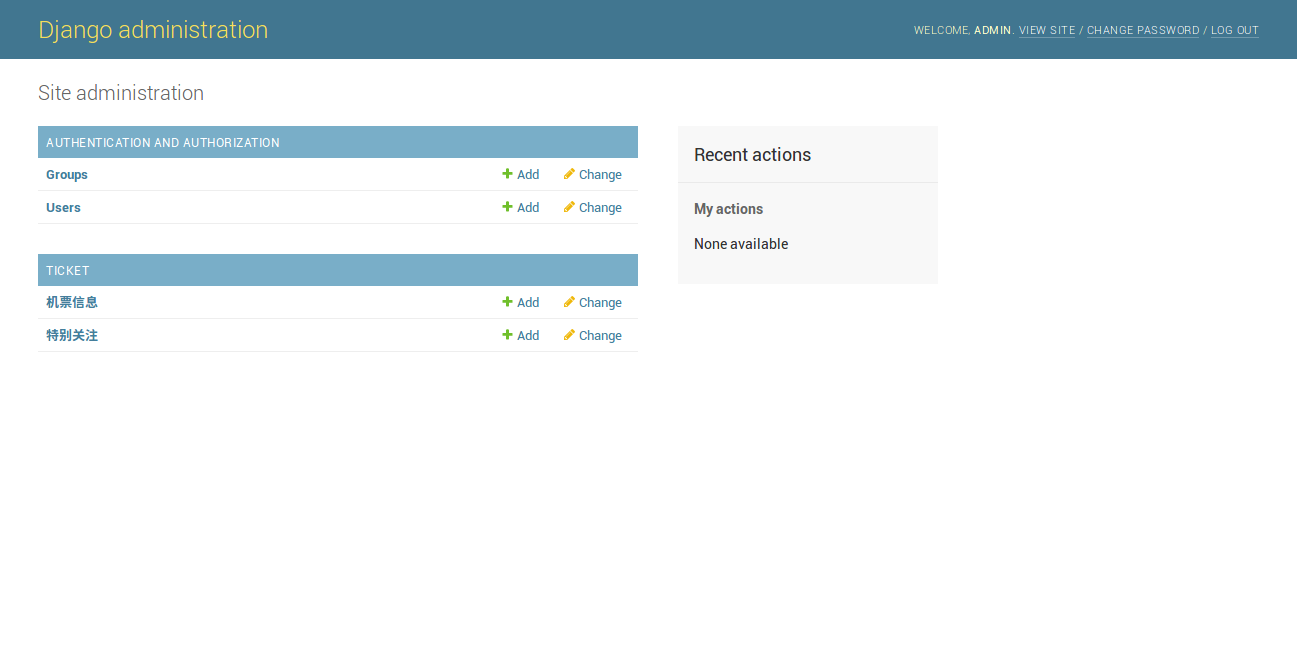


图6－3 管理界面

### 6.3.3 管理员对航班信息的管理

管理员可以对航班信息实现增删改查，其具体实现效果如图6－4所示。

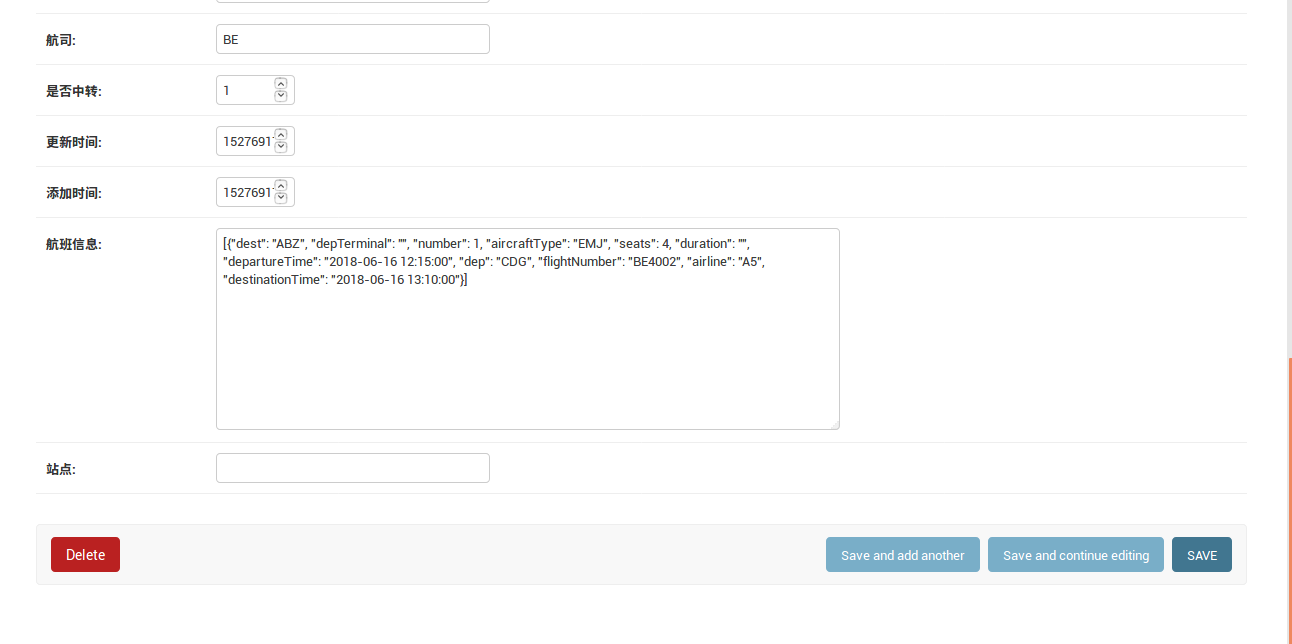


图6－4 修改信息界面

### 6.4 本章小结

本章主要对本系统的代码实现做了详细的分析与展现。在数据获取方面，Scrapy功不可没，数据的完整获取全依赖于它，主要实现获取任务，获取分析并处理数据，最终实现对数据的入库操作。于此同时，分布式的Flask接口部分也必不可少，如果没有它，各个模块之间的交互，数据的共享就无法实现，这部分主要实现了对任务的添加，暂停，恢复，对命令的插入和获取等功能。Web管理后台通过Django的Admin模块来处理，实现对数据的友好展示，友好处理。

# 总结

本文比较详细地介绍了分布式获取机票航班信息并管理数据系统的设计和功能实现，满足了开发的基本要求，所有的模块也都能够按照预期正常地运行。利用该系统可以实现对数据的获取，管理和修改。

在开发本系统的过程中，也走了不少的弯路，最最开始用的是简单的requests来抓取数据并自己写python处理数据的各种语句。感觉也还可以，但是一旦有功能变动，每改一次都要耗费巨大的精力来处理，不方便管理。所谓框架也就这点好处了，轮子帮你处理好，什么功能在哪里写，在哪里实现都有固定的规范，能够做到极大地方便维护和更新。在编写vy爬虫时，并没有接触过分析APP端，只有分析网站，涉及到cookie, 302跳转，以及和token类似的密钥关键字。在我看来，再难的网站也能获取到数据，最终花费了一个月的时间用Selenium模拟人处理浏览器的各种操作，包括填写表单、执行js等。后来看大神的爬虫发现并不只有网站这一个地方可以获取数据，还有一个东西叫移动端，移动端获取数据一般都是通过接口，而且移动端要考虑到用户体验，反爬也不会做得太过分。在APP Store里搜索了一下，果然有，用Charles抓包工具进行分析，携带一个cookie，一个install\_id，经过不断的测试发现果然可以。但是一直用一个cookie和install\_id爬取一天左右就会被封掉，封掉后再过一天左右就又可以获取到数据。在这里采取的方式是提前储存三个cookie和install\_id, 每次用其中的一个，随机抽取。Flask分布式接口部分实现了对redis库的封装，对任务的分发处理等。

后台管理实现了对数据的友好展示处理，这个管理后台也是我做的第一个web项目，还有些许青涩的气息。

本系统也有不足之处，鉴于Python的性能，如果爬虫数量特别多，Flask API可能会承受不了高并发，高流量的冲击。还好现在就两个爬虫，如果爬虫数量增加，可以采用openresty＋lua 重写redis接口部分。当然，这个就是后话了。

该系统是由我自己独立完成的， 在这个过程中我也学到了很多东西，学会了怎么选择适合自己项目功能的框架，怎么对网站的请求进行分析。也学会了如何将理论知识运用到实际项目中。在实际项目中会发生好多理论没有考虑到的各种情况。我相信这个毕业设计会对我以后职业生涯产生不小的影响，尤其是web端还有许多可待优化的地方。未来我一定会更强，它也会更加地完善。

# 致谢

时光如梭，大学四年很快就过去了，大学里面学到了好多东西，大学四年，满满当当的四年青春，遇到过挫折，也享受过成功的喜悦。大学里四年也以这个系统和这篇论文来圆满结束。

首先，要感谢我的毕设指导老师刘洁老师，从开题报告，到中期检查再到毕业论文，一次次地督促，一次次的指导，每一次都给了我很大的感动。老师的敬业精神也深深地触动了我，在一次次殷勤地催促中，我终于完成了此篇论文。

其次，我要感谢我自己选了一个这么有前途且有趣的专业。从刚开始的好奇，到大一的初次接触编程语言C，到后来的大二的JAVA， 以及最终实习学到的Python。一步步把我引进编程的世界，无法自拔，深深地感受到了程序的奇秒。

最后，我要感谢大学里面遇到的所有老师，同学，还有ACM的小伙伴们。老师给了我打开新的世界大门的钥匙，同学们陪我一起成长，一起进步，ACM的小伙伴们起了榜样的作用，让我有了学习路上的模版。一路走来，实属不易，但也是人生中的一笔宝贵的财富。未来，继续加油。

# 参考文献

[1] 宁海元等译.高性能MySQL.电子工业出版社，2013.5

[2] 王海鹏译.Python编程快速上手-让繁琐工作自动化.北京：人民邮电出版社，2016.7

[3] 安道译.Flask Web开发. 北京：人民邮电大学，2015.1.1

[4] 范传辉. Python爬虫开发与项目实践.北京：机械工业出版社，2017.3

[5] 安道;吴珂译.流畅的Python.北京 :人民邮电出版社，2017.5.15

[6] 徐旭铭译.Django Web开发指南.北京：机械工业出版社，2009.5

[7] 金蓓弘译.分布式系统概念与设计.北京：机械工业出版社，2008.1

[8] 李斌译.精通Python爬虫框架Scrapy.北京：机械工业出版社，2018.2

[9] 李子骅.Redis入门指南.北京：人民邮电出版社，2013.5

[10] 黄健弘.Redis设计与实现.北京：机械工业出出版社，2014.6

[11] 李松峰译.Bootstrap实战.北京：人民邮电出版社，2015.5