分类号_____密级 _____

UDC _____

本 科 毕 业 论 文（设计）

分布式数据下载系统设计与实现

学生姓名 黄龑 学号 010022011021

指导教师 刘培顺

院、系、中心 信息科学与工程学院

专业年级 2011 级计算机科学与技术

论文答辩日期 年 月 日

中 国 海 洋 大 学

题目

分布式数据下载系统设计与实现

完成日期：_____

指导教师签字：_____

答辩小组成员签字：_____

_____

_____

_____

_____

_____

_____

分布式数据下载系统设计与实现

摘　要

本文设计了一个支持分布式运行的数据定时下载平台，能够将科研人员的所有下载需求集中到一台或者几台配置高速带宽的数据服务器上面，从而实现集中化自动化管理，减轻科研进程中这些繁琐准备工作所带来的负担。

该系统使用 Python 语言开发，数据库选用 SQLite，以 web.py 作为前端框架，后台支持使用标准库中的 urlretrive 函数以及著名的开源下载工具 aria2、wget 作为下载核心；支持多用户环境，具备灵活的扩容机制以及丰富完善的定时策略，能够最大化地满足多样的实际需求。

关键词：分布式，数据下载，PYTHON，WEB.PY

Design and Implementation of Distributed Data Downloading System

Abstract

This project is an implementation of a distributed data automatic downloading platform, which could be running on data servers with high bandwidth, in order to achieve centralized automated management, reducing the tedious process of data preparation in scientific research.

The system is developed with the Python programming language, uses SQLite as back-end database, selects web.py framework to construct front-end services, and supports to use urlretrive—which is a standard library function, and the famous open-source download tool aria2 and wget as background download core; also supports multi-user environment, with flexible expansion and various timing rules to maximally meet the diverse actual needs.

KEY WORDS： DISTRIBUTED, DATA DOWNLOADING, PYTHON, WEB.PY

目　录

分布式数据下载系统设计与实现

1 需求提出与分析

随着卫星遥感技术的飞速发展，越来越多的卫星观测数据被用于海洋、大气科学的研究中，并取得了明显的效果，极大地促进了科研的进展。但是由于观测数据的精密度不断在提升，数据规模与体积越来越大，因此在应用这些数据进行研究的过程中，首先要解决的问题就是，如何便捷地获取到相关单位所公布的数据。由于一些网络原因，直接下载国外网站上所提供的数据是比较耗时的，并且也没有必要重复地下载这些体积庞大的数据。另一方面，很多卫星数据是自动定时发布的，由人工去一个一个下载显然是十分繁琐的事情。为了解决这些问题，本文设计了一个支持分布式运行的数据下载平台，能够将科研人员的所有下载需求集中到一台或者几台具有高速带宽的数据服务器上面，从而实现集中化自动化管理，减轻科研进程中这些准备工作所带来的负担。

1.1 需求分析

作为一个能够解决用户实际需求的定时数据下载系统，本项目需要具备以下这些功能，并且保证代码实现的简洁可靠，避免出现意外的运行时故障。

1) 支持多用户环境，分布式运行；
2) 支持任务定时下载，以及多种定时方式；
3) 支持根据日期、时间的变化来生成下载地址；
4) 支持按照不同时区来制定自动下载任务；
5) 支持多线程下载，能够选择不同的下载工具；
6) 支持任务完成状态检查，以及失败后的重试机制；
7) 支持下载文件按照目录分类保存，以及可自定义的文件命名方式；

下面将分析上述这些需求的必要性，以及实际的使用场景。

由于科研过程中所用到的各类数据来源多种多样，有着不同的定时发布周期：有的按天发布，有的按月发布等等，因此必须实现一种灵活的定时策略，支持任意方式的定时下载。而对于定时发布的数据，常见的一种情况是其下载地址随着日期变化，如下面这个卫星数据文件名：

S201003250600.L3m_MO_CHL_chlor_a_9km

在它的文件命名规则中，包含了可变的年份、月份、日期以及小时。因此，为了能够自动下载这样的文件，本项目也应该支持指定对应的自动变量，从而实现在到达设定的时间时，根据 URL 规则来生成真实的下载地址。考虑到发布数据的机构分布在世界各地，他们的发布时间显然会按照自己的当地时区计算，为了避免用户在每次添加定时下载任务时，先去查发布者所在时区，再换算时间的繁琐，本项目

支持添加规则时选择对应时区，从而大大简化了操作流程。

数据文件的体积一般会比较大，再加上由于我国网络部署的缺陷，导致访问外国的数据网站尤其缓慢。在这样的背景下，为了能够充分利用数据下载平台的高带宽，下载平台必须能够支持多线程下载。但另一方面，很多数据提供方的服务器又设置为只支持单线程连接，于是本项目需要设计为能够灵活切换下载方式，从而方便使用。

为了方便数据文件的分类存储与共享，用户在提交任务时需要能够选择下载的保存位置，以及文件命名规则。这样，就可以方便用户之间，或者管理员向所有用户共享数据，避免重复提交相同的任务。

最后还需要考虑的是，数据下载完成后的状态检查问题。很多时候，数据发布方并不是自动发布数据，而是人工处理，这就导致了数据的真实发布时间并非精准且固定的。为此，必须设计一套状态检查机制，用于检测数据下载是否成功，以及在发现失败后自动重试下载。

1.2 可行性分析

在上述需求分析中，与定时下载策略相关的部分，必须自行实现；但其余的很多需求已经有了成熟且稳定的开源实现，可以直接用作本项目的基础组件。例如，开发一个完善稳定的高性能多线程下载工具是比较复杂的任务，需要考虑网络传输过程中的种种细节，全都实现一遍的代价非常大。但是现在已经有了跨平台的 wget、aria2 下载器，完全可以直接应用于本项目作为下载后端，不仅节约了开发成本，而且在性能上也要明显优于 Python 自带的 urlretrive 等标准库。为了检查数据文件是否下载成功，需要一个可靠的文件类型检测工具，来检查下载到的是否是用户想要的文件。如果从头来实现这样一个工具，更是需要巨量的繁琐工作，但是跨平台的 file 命令结合 Python 的 subprocess 模块，可以直接用于文件类型的检查，并且能够支持几乎所有的文件类型。综上所述，本项目的需求分析中一些不大容易实现的部分，都已经有了成熟可靠的开源解决方案，因此本项目在设计上是完全可行的。

另一方面，考虑到用于科研的高性能计算集群，一般具有“管理节点+计算节点+全局后端存储”的配置模式，并且计算节点多数情况下会有闲置，十分适合在上面部署分布式数据下载平台，从而充分利用计算资源以及带宽资源，也方便科研人员直接在集群上处理数据、运行模式。因此，本项目也有着实际且具体的应用场景。

2 技术选型

2.1 语言选择

本项目的主要使用者为科研人员，一般不具有较高的计算机操作水平与编程经验，考虑到具备高带宽的数据下载服务器可能使用 Windows/Linux 操作系统，为了保证这类用户能够顺利将项目部署在不同的平台上，则需要本项目在开发完成后，能够具有这样几个特性：跨平台运行、无需编译、依赖关系简单、打包/迁移方便。经过考察后，发现著名的 Python 语言恰好能够满足这样的需求。

Python 是一种面向对象、命令式的计算机脚本语言，也是一种功能强大的通用型语言，已经具有近二十年的发展历史，成熟且稳定。它包含了一组完善而且容易使用的标准库，能够轻松完成各类常见的任务。Python 支持命令式编程、面向对象程序设计、函数式编程、面向切面编程、泛型编程多种编程范式，并且与 Scheme、Ruby、Perl、Tcl 等动态语言一样，具备垃圾自动回收功能，能够自动管理内存空间。Python 经常被当作脚本语言用于处理各类系统管理任务和 web 编程，然而它也非常适合完成各种高阶任务，如混合编程，分布式数据处理，高性能计算等等，尤其适合作为一种胶水语言，将系统底层的高性能部分，与上层的业务逻辑隔离开来，大大提升了开发效率。Python 虚拟机本身几乎可以在所有的操作系统上面运行，使用一些诸如 py2exe、PyPy、PyInstaller 之类的工具可以将 Python 源代码转换成可以脱离 Python 解释器运行的程序，而 virtualenv 之类的辅助模块也可以轻松将复杂的 Python 运行环境打包发布。综上所述，本文选取 Python 语言作为主要的开发语言。

2.2 数据库

根据需求分析可知，本项目的应用范围，一般是处于共同的校园网/局域网环境下的科研人员。因此，这也就限定了在项目的实际应用环境下，数据库中不会存放大量的用户信息或者下载规则，读取/写入负担不会太大。同时考虑到本项目必须具有方便部署的特性，而 MySQL 之类的常用数据库系统一般都需要单独配置，因此，本项目最终选择了简单小巧、资源占用率低的 SQLite 数据库来存储用户数据以及运行时状态。

SQLite 是一个开源的嵌入式关系型数据库，实现了一个自包容、零配置、支持事务的 SQL 数据库引擎。其特点是高度便携、使用方便、结构紧凑、高效、可靠。与其他复杂数据库管理系统不同的是，SQLite 的安装和运行非常简单，在大多数情况下，只要确保 SQLite 的二进制文件存在，即可开始创建、连接和使用数据库。SQLite

的这些特性恰恰能够满足本项目方便部署、迁移，依赖关系简单的需求。

2.3 Web 框架

Web 应用框架，或者简称“Web 框架”，实际上是建立 web 应用的一种方式。从简单的动态博客系统到复杂的富 Ajax 应用，互联网上的每一个页面都是通过写代码来生成的，但是这些代码之间存在大量相似的部分。为了将开发者从繁琐且无意义的重复性劳动中解脱出来，web 框架的概念应运而生。从本质上来看，web 框架实际上就是向程序员隐藏了处理 HTTP 请求和响应请求相关操作的基础代码架构。至于隐藏多少信息，则取决于框架设计者的思路。例如，著名的 Django 框架和 Flask 框架走向了两个极端：Django 试图囊括 web 应用开发中的一切场景；而 Flask 则立足于“微框架”的设计哲学，仅仅实现 web 应用需要的最小功能子集，其他功能则交给开发者自行解决。

Web.py 是一个轻量级 Python web 框架，它简单而且功能大。该框架由美国作家、Reddit 联合创始人、RSS 规格合作创造者、著名计算机黑客 Aaron Swartz 开发，并且目前已被很多家大型网站所使用。Web.py 的设计理念力求精简(Keep it simple and powerful)，自身的代码量不多，不像 Django 之类的重量级 web 框架那样，坚持提供全套的解决方案，而是只提供的一个框架所必须的少数功能，如：URL 路由、HTML Template、数据库访问等等，其它细节则交给用户自行定制。一个框架精简的好处在于可以让开发者将精力聚焦在业务逻辑上面，而不用付出太多的时间成本来学习框架本身的特性，或受到框架设计思路的干扰。但是其缺点也很明显：对于很多需求没有提供一站式的解决方案，需要开发者自行设计。考虑到本项目主要解决的是多用户环境下的定时数据下载需求，业务逻辑更多地聚焦在如何实现稳定、可定制的下载调度器，而非 web 页面上，因此选取轻量级的 web.py 来作为项目所基于的 web 框架。

2.4 HTML 渲染模版

Web.py 框架自身提供了一套非常精巧的基于 Python 语法的 HTML 模版系统，具有强大且完善的功能。但是通过实际试用后发现，它的语法定义在很多细节上十分繁琐，比如复杂的转义机制，与习惯相悖的符号规则，以及非常别扭的语法缩进。这些与开发者习惯不符之处，很容易引发隐晦的 BUG。但由于 web.py 自身定位是一个精简版框架，它并不限制开发者只能使用其自身提供的模版解决方案，因此，通过比较与试用，本项目最终选择了 Mako 模版系统来作为编写 web 前端的工具。

Mako 是一个高性能的 Python 模板库，它的语法和 API 借鉴了很多其他流行 HTML 模板的精华特征，如 Django、Jinja2 等等，但又有着其自身的优势。Mako 模

板文件可以包含 mako 所定义的指令，如：变量、表达式、控制结构（条件控制、循环控制）、服务端注释、Python 代码以及各种 HTML 标签等等，所有这些字符流最终都会被编译成 Python 代码，从而获得较高的渲染性能。从本质上来讲，Mako 模版系统就是一种嵌入式的 Python 语言扩展，它通过改进组件化布局和继承的思想，构建出一种简明且灵活的 HTML 描述模型，并同时做到了贴近 Python 语言的语义限定。上述特性非常有助于本项目的开发，实现了 web 前端与后台下载系统的无缝结合。

2.5 第三方依赖库

下面简单介绍对于本项目而言的三个重要依赖库，这些依赖库能够以非常简洁优雅的方式，来解决本项目所面临的几个核心需求。

2.5.1 virtualenv

尽管 Python 是支持跨平台运行的编程语言，但是在运行 Python 程序前必须要先装好相关的运行环境，即 Python 解释器，以及第三方依赖库。这对于没有相关编程经验的用户而言，是较难顺利完成的任务。并且，并非所有用户都具有数据下载服务器上的最高权限（对于 Windows 而言是 Administrator，对于 Linux 则是 root 账户），不一定有权随意安装软件。因此，本项目有必要提供一种简单可依赖的方式，用于快速配置好下载服务器，并上线运行。

Python 的 virtualenv 模块是解决这个需求的一大利器，它用于创建并启动独立的 Python 环境，使得多个 Python 解释器及其依赖库相互独立，互不影响。利用 virtualenv 能够实现下述三方面的功能：

1) 在没有权限的情况下安装并运行 Python 解释器；
2) 不同解释器可以使用不同的模块版本；
3) 模块的升级不会影响其他应用的运行。

从某种意义上而言，virtualenv 相当于能够任意创建并封装独立的运行环境。这样做的好处一方面在于，可以完全避免开发者在开发项目时所用的依赖库，与用户所安装的依赖库之间的版本兼容性问题，确保了只要开发者能够正常运行该项目，则在用户处也不会出现问题；另一方面也方便了本项目的打包与安装——在用户处只要解压出打包好的运行环境，即可运行该项目。

2.5.2 requests

应用单元测试技术能够对本项目进行自动化的测试，从而减少代码逻辑中出现 BUG 的概率，同时也方便他人协同开发，以及自动化合并代码分支。对于本项目这样具有前端页面的 web 项目而言，为了能够方便快捷地撰写测试用例(test case)，则

需要选取一个适合的 HTTP 库，来模拟浏览器的 GET/POST 行为，从而实现自动化测试。尽管 Python 语言默认提供了这样的标准库即 urllib1-3，但是这些库的 API 被设计得极为不友好，使用它们时需要繁琐的工作，甚至包括各种方法覆盖，来完成最简单的任务。因此，我们不得不将目光投向其他更好用的第三方 HTTP 库。

Python requests 库的全称是“Requests: HTTP for Humans”，由此可见该库的开发者对于其易用性充满了信心。Requests 库的实现本质上是基于 urllib3，因此继承了它的所有特性：支持 HTTP 连接保持和连接池，支持使用 cookie 保持会话，支持文件上传，支持自动确定响应内容的编码，支持国际化的 URL 和 POST 数据自动编码，等等。这些功能为项目开发中的单元测试环节，提供了最为便利的条件。

2.5.3 datetime

在“需求分析”一节中已经提到，本项目需要支持针对不同时区的定时下载任务。为了处理时间的转换关系，则需要一个功能强大的日期时间处理库来作为支持。Python 的 datetime 模块，恰好就提供了这样的功能，它内置了各个时区的相关信息，能够以简洁优雅的方式自动处理时间日期的转换，甚至日期之间的加减，从而使得实现一个简单可依赖的分时区定时下载机制成为了可能。虽然 SQLite 数据库不包含原生的日期时间存储格式，但是 datetime 模块的著名外部模块 dateutil 提供了一组强大的 parse 方法，能够智能识别字符串中的时间日期信息，并转换为对应的 datetime 对象。于是这样便可以解决如何将日期时间信息存入数据库的问题。

3 系统设计与实现

在介绍本项目的整体设计思路之前，有必要引入一个“实例化”的概念，用以方便后面的阐述。先前已经提到，对于定时发布的数据，其下载链接一般是按照一定规律来变化的。因此，用户在输入下载链接时，一般可以认为是在制定生成链接的“规则”，而不是一个固定的链接本身。对于本项目而言，这种规则首先可以是固定的链接地址，也可以是某种带有自动变量的 URL 链接，甚至可以是用来生成链接的 Python 表达式。但是系统在实际发起一个下载任务的时候，必然是将一条抽象可变的规则，转换成为一个具体的下载链接。因此，本文将这一转换的过程，称为一次实例化。

3.1 整体架构

作为与用户直接交互的 web 前端，其主要职责是以简洁明了的方式获取用户输入，并反馈给用户信息。为此，本项目采用了前端与后端分离的设计理念，其结构

示意如图 1 所示。这样的设计一方面保证了系统的稳定性，另一方面也有助于分离主下载服务器与从下载服务器，降低系统的耦合程度以及冗余程度。

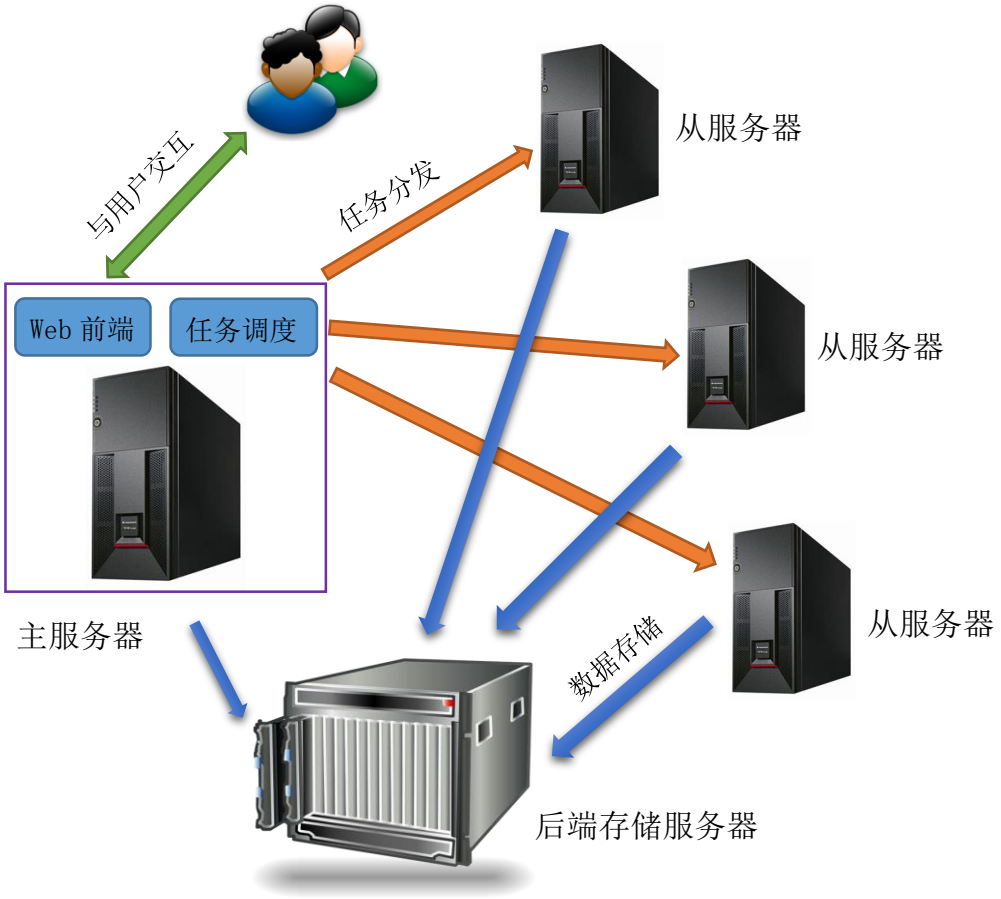

Figure 1 系统结构设计图

图中的彩色箭头代表着数据/信息的流动，从中可以看出，主下载服务器在整个系统的拓扑结构中起到至关重要的作用：一方面，它是连接用户与系统本身的唯一桥梁，承担着重要的交互职责；另一方面，它也是下载任务调度和分发的中心，通过这台服务器，实例化的任务被各自分发到对应服务器上，从而完成下载任务的执行。因此，在下载任务量大的情况下，为了保证主服务器能够有充分的计算资源去处理前端渲染以及后台定时逻辑等任务，应该为其设置较小的下载任务负载，即开启较小的线程池。

此外，如今已经有大量成熟可靠的方案用于实现全局共享存储，比如 Windows 平台下的 Samba 协议，Linux 平台下的 NFS 协议等等。为了方便部署以及与服务器系统的整合，本项目不考虑设计新的分布式存储体系。

3.2 数据库设计

数据库是连接前台用户操作与后台系统运行的纽带，其设计的好坏直接关系到

系统能否正常稳定地运作。由于本项目所处理的业务逻辑并不十分复杂，因此具有相对清晰简明的数据库结构。下面详细介绍数据库中三个基本表的属性。

- 用户数据表：用来保存数据下载系统使用者的个人信息以及登录信息。此外，用户资源配额也存放在数据表中，管理员可以设置用户最多能够占用多少存储空间/下载多少数据文件。

表格 1 用户数据表

字段名	类型	功能
UID	TEXT	存储用户的全局唯一标识符（主键）
SessionID	TEXT	存储用户的登录 session
UserStatus	INTEGER	标识用户状态（管理员/正常/被封禁）
UserName	TEXT	存储用户真实姓名
PassWord	TEXT	存储用户密码（暂不考虑安全问题）
Tel	TEXT	用户联系方式
E-mail	TEXT	用户联系方式
MaxSize	INTEGER	最大可使用的空间
MaxFiles	INTEGER	最大可下载的文件数量
Downloader	TEXT	设置默认下载器
NameRule	TEXT	设置默认文件命名规则

- 下载规则表：存储下载任务的 URL 生成规则以及附加的相关信息。用来精细控制每一条规则的具体执行细节。

表格 2 下载规则表

字段名	类型	功能
TaskID	INTEGER	下载规则的全局唯一标识符
UID	TEXT	该规则所属的用户 UID
URL_Rule	TEXT	以字符串形式存储 URL 的生成规则
Rule_Name	TEXT	规则名称
RepeatType	INTEGER	定时重复类型

RepeatValue	TEXT	将定时运行所需要的信息压缩在字符串中
TimeZone	TEXT	下载任务所属时区
Status	INTEGER	规则状态（启用/停用）
SubDirectory	TEXT	数据文件保存到的位置
NameRule	TEXT	文件下载好后的命名规则
TaskTime	INTEGER	任务的最大重试时间（超时后放弃重试）
Downloader	TEXT	任务所使用的下载器
CheckType	TEXT	下载结束后，检查是否正确完成的方式
CheckSize	INTEGER	以文件大小来判断的话，最小的文件体积

- 任务运行状态表：存储当前正在运行的任务的相关信息。这些信息其实可以保存在程序的运行时结构中，但在本项目中却将其存入数据库。这样的设计一方面是考虑到，正如前文所述，本项目的数据库读写负载相对比较小，能够承担得起这样的“性能浪费”；另一方面更重要的原因在于，这样的设计保证了即便程序运行中途意外崩溃，在被守护进程（如 Linux 平台上的 supervisord）短时间内重启后，不会丢失先前生成的运行时信息；甚至从用户角度来看，程序相当于没有出现故障。

表格 3 任务运行状态表

字段名	类型	功能
UID	TEXT	记录当前任务所属的用户 ID
URL	TEXT	记录当前下载任务的真实 URL
Status	INTEGER	任务运行的当前状态（就绪/运行中/失败）
Location	TEXT	文件保存的绝对路径
StartTime	TEXT	任务起始时间
FinishTime	TEXT	任务超时时间
TaskID	INTEGER	任务所属的规则 ID
TimeZone	TEXT	任务所属的时区
RepeatTimes	INTEGER	任务已经重复的次数

数据库结构的设计大体如上所示。从更为具体的角度来讲，先前所定义的“实例化”，实际上就是将“下载规则表”中的规则内容转换（解释）后，写入“任务运行状态表”的过程。基于这样一个简单精简的信息存储体系，本项目实现了稳定可靠且功能强大的定时下载机制，将在接下来的一节进行深入介绍。

3.3 定时下载策略

本项目支持下载任务按照年、月、周、日为周期的定时重复，以及仅执行一次的定时下载。考虑到很少有数据以小时为间隔发布，因此不提供按小时执行任务的选项。即便真有这样的需求，也可以通过添加 24 条以日为周期的定时规则来实现。

为了实现相对复杂的定时逻辑，本项目将执行一次下载任务的过程具体细分为三个阶段：

1) 下载规则实例化为下载任务；
2) 下载任务被添加到线程池缓冲区；
3) 缓冲区中的下载任务被线程取出，并执行下载。

3.3.1 规则实例化

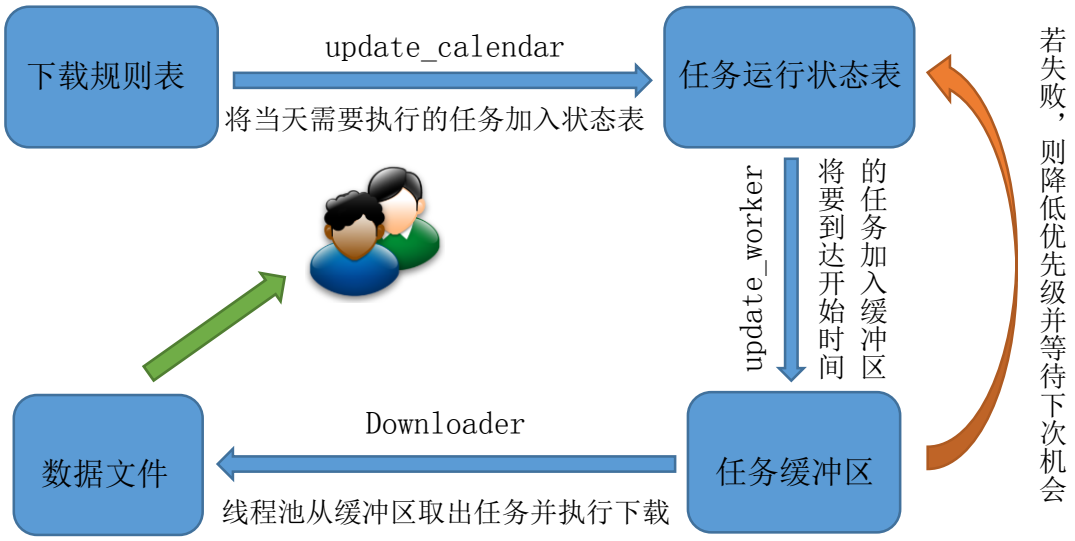

Figure 2 规则的执行流程

如图 3 所示，由于系统的最小定时粒度是“天”，因此在新一天的开始，系统需要读取所有的下载规则，并将当天需要执行的规则实例化成下载任务，加入“任务运行状态表”，这项工作是由一个独立的线程“update_calendar”来完成的。该表中的任务按照失败重试次数从低到高来进行排序，下载失败次数多的任务会被设置

为较低的优先级，以免影响到正常任务的执行。一个重要的细节是如何判断进入了新的一天：由于不同时区进入新一天的时刻各不相同，为了保证每个时区的任务都能够被正确执行，需要“update_calendar”线程每隔一段时间（约 10 分钟左右），就检查一次是否在某个时区已经到了新的一天，从而及时开始实例化下载规则。

上述实例化过程只是将当天应该执行的任务放入了数据库中，并附带了该任务具体的开始时间和超时时间。但是这个任务不应该立刻就被放入缓冲区被执行，否则无法正确下载到数据，只会不断重试，占用系统资源。因此，需要设置一个“update_worker”线程每隔一段时间（约 10 秒左右）扫描一遍当天要执行的所有任务，即“任务运行状态表”，并进行处理：如果有任务已经到达了开始时间，或者上次下载失败，就将其加入线程池缓冲区；如果有任务已经超时但仍未成功下载，就将其移出数据表。

最终，缓冲区中的任务被下载线程取出并执行下载，保存到用户所设置的位置，即完成了一次任务执行流程。

3.3.2 线程池的应用

本项目中的线程池和下载线程类被单独封装为一个模块，考虑到项目的设计理念是精巧而非通用，因此放弃了模版元编程的思想，而是简单实现一个能够满足基本需求的线程池。它的任务分发方式基于经典的“生产者-消费者”模型，最为重要的部分是“insert”和“remove”方法，用于向任务缓冲区插入元素，以及从任务缓冲区读取元素。其核心代码如下所示：

```
    def insert(self, data):
        self.slots.acquire()
        self.mutex.acquire()
        self.working_queue.append(data)
        self.mutex.release()
        self.items.release()
    def remove(self):
        self.items.acquire()
        self.mutex.acquire()
        result = self.working_queue.popleft()
        self.mutex.release()
        self.slots.release()
        return result
```

其中的“working_queue”即为任务队列；items 和 slots 是两个信号量，用来对可用资源计数；mutex 则是一个互斥锁，用于保证对任务队列进行操作时的原子性。

3.4 UI 设计

先前已经提到，本项目 web 前端的主要功能是与用户进行交互，在实现时被简单地拆分为一个独立的 DownloadServer.py 文件用来提供数据接口，以及一些用于渲染后返回给用户的 HTML 模版。这些模版主要分为以下几大功能：

1) 登陆管理
2) 用户管理
3) 系统日志
4) 系统设置
5) 用户自定义规则
6) 下载任务状态查询

这样，本项目可以通过仅仅修改 HTML 模版，来实现对于前端样式的修改；并且只需要为后台数据接口写单元测试——事实上页面前端的多变性使得它在多数情况下只能人工测试。此外，诸如字段检查、自动提示之类方便用户使用的功能，也以 CSS 结合 JavaScript 的方式编码在前端页面里。因为该系统并不是一个以复杂交互为特点的项目，因此这样精简的设计模式已经足以实现需求目标。

最终，本项目实现的前端页面显示效果如下组图所示：

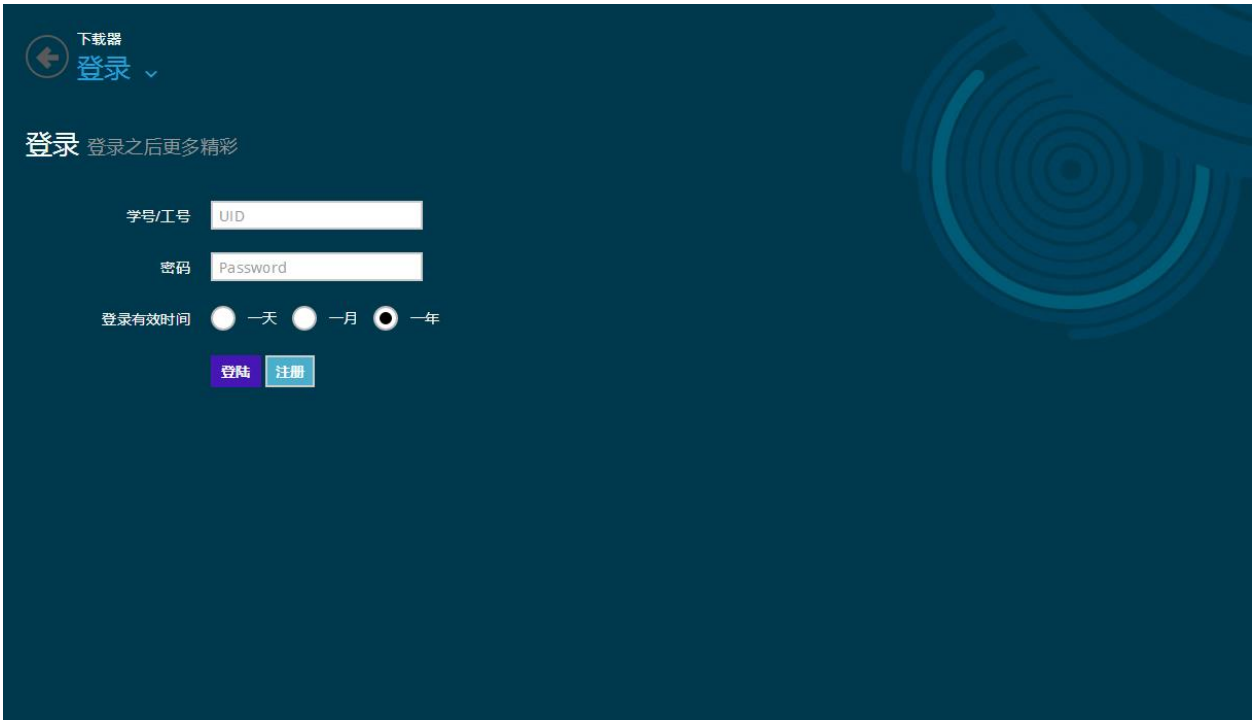

Figure 3 登录页面

Figure 4 平台主界面

Figure 5 系统管理页面

具体效果可以访问 http://graduation-project.finaltheory.info 进行体验。

4 单元测试

单元测试(unit testing)，是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般而言要根据实际情况去判断其具体含义，如 C 语言中单元指一个函数，Java 里单元指一个类，Fortran 语言里则是指一个子程序或者函数。总而言之，单元就是人为规定的最小的被测功能模块。单元测试是在软件开发过程中要进行的最低级别的测试活动，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试，从而保障整体系统的可用性与可靠性，同时也保证了后续对代码的改动不会破坏现有的功能。

本项目使用 Python 标准库中的 unittest 模块来进行单元测试，主要工作是利用 requests 等 HTTP 库，来模拟浏览器行为，检查数据接口的行为以及下载系统的行为是否符合预期，即：是否完成了预期的数据操作（如增加/删除规则、用户/系统管理），是否正常将下载规则实例化、实例化后的下载任务是否被正确地取出并加入缓冲区、缓冲区中的下载任务是否能够被正确地执行，配额管理之类的机制是否正确运行等等。

上文已经提到，本项目当前只针对后台服务端的数据接口以及下载任务的处理逻辑进行测试。具体做法是，首先定义一个继承自 unittest.TestCase 类的测试类，命名为“BasicTest”，这个类中包含了所有的测试方法。接下来，为这个类定义全局的初始化和清理函数（并非构造和析构函数），用于进行测试开始前的准备工作，如备份数据库和配置文件等；以及测试完成后的清理工作，如删除临时文件等。最后，在这个测试类中实现对于项目各个功能的测试，这些测试代码的大体形式如下所示：

```
def test_03_add_rule_03(self):
        s, r = self.login('test02', '12345678')
        # 确认正确登录
        self.assertTrue(u'管理规则' in r.text)
        post_url = 'http://localhost/modify_rules'
        post_data = data_modify_rule
        # 增加一个月任务
        tz = 'Asia/Shanghai'
        test_name = u'测试03'
        now = datetime.now(timezone(tz))
        post_data['RepeatType'] = 'month'
        post_data['day'][0] = str(now.day)
        post_data['hour'][2] = str(now.hour)
        post_data['minute'][2] = str(now.minute)
        post_data['Rule_Name'] = test_name
        post_data['Sub_Dir'] = test_name
        post_data['TimeZone'] = tz
        r = s.post(post_url, data=post_data)
        # 检查操作是否成功
        print r.text
```

```
            self.assertTrue('200' in r.text)
            # 检查是否生成了对应的目录
            self.assertTrue(os.path.exists(os.path.join(temp_dir, 'test02',
        test_name)))
```

本项目的测试样例最终覆盖了接近 80%的项目代码，从而使得我们在将其应用于具体业务时，能够对其可靠性有着充分的信心。

5 总结与改进

通过设计严谨的执行逻辑，撰写完善的单元测试，本文实现了一个功能强大且简单可依赖的分布式数据下载平台。但是依然不得不承认，本项目在整体设计模式上存在着一些不足之处。

首先是作为支持分布式运行的系统，本项目暂时不具备完善的灾难恢复逻辑。在系统结构分析中已经提到，所有从下载服务器的运行依赖于主服务器的正常运行，如果它出现了意外故障，则会导致所有任务无法继续进行。考虑到从服务器与主服务器只是运行模式的不同，因此完全可以实现一个自动的故障检测机制，即如果从服务器检测到主服务器掉线，则自动将自身切换为主服务器。这也就是著名的“去中心化”思想。不过，实现这样的机制，需要一个合适的数据库同步策略，将主服务器的所有数据信息同步到所有从服务器上面，本项目在这方面暂时没有做出完善的设计。

另一方面，在分布式运行数据下载平台时，本项目暂未设计出一个优雅的任务分发机制。在当前的设计中，主服务器通过 POST 方法向从服务器发送下载任务，这就要求主服务器端必须保存所有从服务器的地址列表，给系统的部署增添了些许繁琐。似乎尚未有可行的解决方案能够弥补该设计的不足之处。

参考文献

[1] Chun W J, 吉广. Python 核心编程[M]. 人民邮电出版社, 2008.
[2] Lutz M. Learning python[M]. " O'Reilly Media, Inc.", 2013.
[3] van Rossum G, Drake F L. Extending and embedding the Python interpreter[M]. Centrum voor Wiskunde en Informatica, 1995.
[4] van Rossum G, de Boer J. Interactively testing remote servers using the Python programming language[J]. CWI Quarterly, 1991, 4(4): 283-303.
[5] 曾浩. 基于 Python 的 Web 开发框架研究[J]. 广西轻工业, 2011 (8): 124-125.
[6] Grehan R. Pillars of Python: Web. py Web framework[J]. InfoWorld IDG Retrieved January, 2013.
[7] Kristensen A. Template resolution in XML/HTML[J]. Computer Networks and ISDN Systems, 1998, 30(1): 239-249.
[8] 周一丁. 基于面向服务并行计算的 Python 计算网格[D]. 上海交通大学, 2008.
[9] 秦啸, 韩宗芬, 庞丽萍. 基于异构分布式系统的实时容错调度算法[J]. 计算机学报, 2002, 25(1): 49-56.
[10] 胡华平, 金士尧. 分布式系统可靠性模型[J]. 计算机工程与应用, 1999, 35(8): 1-3.
[11] Gioachin F, Kalé L V. Dynamic high-level scripting in parallel applications[C]//Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE, 2009: 1-11.
[12] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python[J]. The Journal of Machine Learning Research, 2011, 12: 2825-2830.