**分布式爬虫系统需求规格说明书**



编写：郑毅 日期：2017.4.17

**文件变化记录单**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 版本编号 | \*变化  状态 | 简要说明 | 变更人 | 变更日期 | 批准人 | 批准日期 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

\*变化状态：A——增加，M——修改，D——删除

**文件批准单**

|  |  |  |
| --- | --- | --- |
| **职务** | **签字** | **日期** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 引言

随着国际互联网的迅速发展，网上的信息越来越多，全球网页数量超过20亿，每天新增加730万网页。要在如此浩瀚的信息海洋里寻找信息，就像“大海捞针”一样困难。在实际生活中我们经常会使用像百度、Google这些搜索引擎检索各种信息，搜索引擎正是为了解决这个问题而出现的技术，而爬虫正是搜索引擎所需要的关键部分，既然百度、Google这些搜索引擎巨头已经帮我们抓取了互联网的大部分信息，为什么还要自己写爬虫呢？因为深入整合信息的需求是广泛存在的，在企业中，爬虫抓取下来的信息可以作为数据仓库多维展现的数据源，也可以作为数据挖掘的来源，甚至有人为了炒股，专门抓取股票信息。

爬虫系统，是对海量的分散的互联网数据进行采集的系统，是搜索引擎系统的基础。大数据近年来快速发展，炙手可热，不仅是数据的容量大，更是强调对全样本的数据的分析。互联网数据中包含了大量有价值信息，是大数据的重要数据来源。

分布式技术不仅可以解决IT运营的成本，还可以解决爬虫效率问题，尤其是当今云计算的热潮，更是关注分布式爬虫系统的效率问题。

## 编写目的

文档展示了整个分布式爬虫系统的整体网络结构和功能结构的概貌，从总体架构上展示出整个系统的轮廓，然后又对功能需求、性能需求和其它非功能性需求进行了详细的描述。其中对功能需求的描述采用UML的用例模型方式，给出了非常直观的用例图。这些文字和图形都为了本文档能详细准确地描述用户的需求，同时也为用户更容易地理解这些需求的描述创造了条件。

该文档详尽说明了这一软件产品的需求和规格，这些规格说明是进行设计的基础，也是编写测试用例和进行系统测试的主要依据。同时，该文档也是用户确定软件功能需求的主要依据

## 文档约定

本文档按以下要求和约定进行书写：

（1）页面的左边距为2.5cm，右边距为2.0cm，装订线靠左，行距为最小值20磅。

标题最多分三级，分别为黑体小三、黑体四号、黑体小四，标题均加粗。

（3）正文字体为宋体小四号，无特殊情况下，字体颜色均采用黑色。

（4）出现序号的段落不采用自动编号功能而采用人工编号，各级别的序号依次为（1）、1）、a)等，特殊情况另作规定

## 预期的读者和阅读建议

该分布式爬虫系统主要包括：综合描述、功能需求、非功能性需求和外部接口描述。综合描述部分主要对系统的整体结构进行了大致的介绍；功能需求部分对系统的功能实现进行了详细描述，是本文的主要部分；非功能性需求部分对非功能需求进行了详细的描述；外部接口需求部分对用户界面、软件接口、硬件接口和通讯接口等进行了描述。

预期读者：小组开发人员，大赛测试人员，用户。

阅读建议：

（1）小组开发人员：

重点关注功能需求和数据字典，对系统的模块有个清晰的认知。

（2）大赛测试人员：

关注数据处理的算法和功能的实现。

（3）用户：

阅读使用说明，能正常使用。

## 产品的范围

分布式爬虫系统实现高速稳定的下载，解决反爬虫问题，着重在URL的去重算法和分布式的调度算法，实现网页自动结构化。该产品主要用于电商网站的爬取工作，给定特定的电商网站实现爬取工作。实现一定的电商通用性，实现对京东，淘宝，天猫，网易严选等网站的爬取。

## 参考资料

无

# 综合描述

分布式爬虫就是使用成千上万台小型机和微机进行合作，完成分布式抓取工作。本系统用云服务模拟小型机进行分布式的抓取工作。

## 产品的前景

随着互联网的高速发展，在互联网搜索服务中，搜索引擎扮演着越来越重要的角色。网络爬虫是搜索引擎系统中十分重要的组成部分，它负责从互联网中搜集网页，这些页面用于建立索引从而为搜索引擎提供支持。面对当前极具膨胀的网络信息，集中式的单机爬虫早已无法适应目前的互联网信息规模，因此高性能的分布式爬虫系统成为目前信息采集领域研究的重点。

## 产品的功能

首先给出数据流图的顶层图如图2.2-1所示：

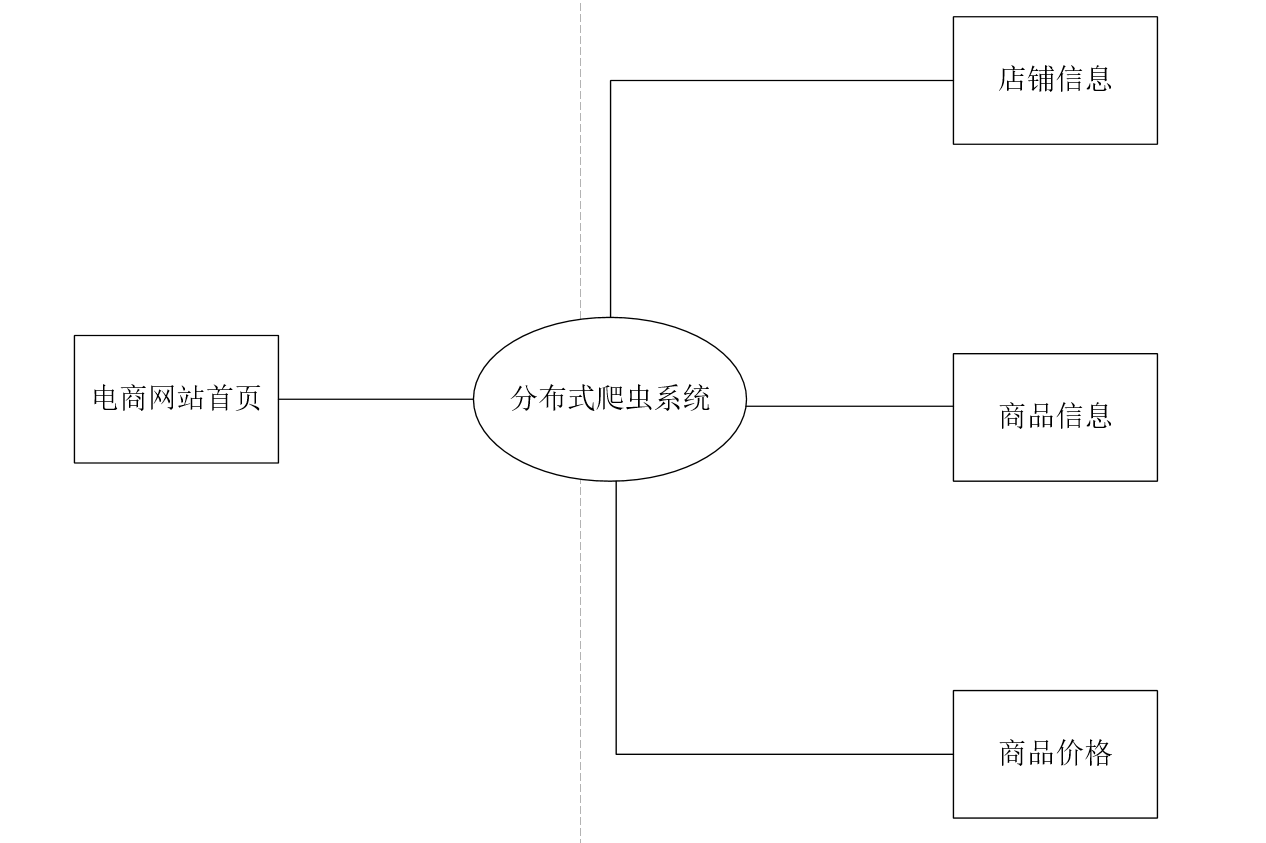


图2.2-1 分布式爬虫顶层数据流图

从顶层数据流图可以看到分布式爬虫系统的基本功能：输入电商网站，得到电商网站中店铺信息，商品信息，商品价格，并在前端web界面显示数据，以及当前工作各个分爬虫机的cpu占用率和内存占用率，用动态图显示整个系统抓取的URL总量。下面是分布式爬虫系统在实现上的一些功能描述：

（1）爬虫策略

爬虫策略，应该保证爬虫的下载快速和高效，能解决爬虫面临的反爬虫问题。输入入口URL之后，自动分析网页的组织形态获取新的链接，进行下载。例如输入XX电商主页地址后，自动分析导航菜单，自动分析翻页地址，自动分析详情页的地址等。

（2）URL去重算法

对URL进行去重，已经下载过的，没有进行数据更新的，不再进行下载。去重算法应考虑内存的问题，内存越小越优。

（3）分布式调度算法

爬虫任务，可以理解为对一个网站的一次采集过程。

分布式爬虫将所有任务在多台机器上分布式执行（可用多进程模拟）。分布式调度策略，应该将不同网站的URL混合后，分配到多台机器上执行。分布式调度策略的重点在URL的分配策略、失败处理等。

分布式调度应该有多种调度策略，满足不同的场景需求。例如，有的任务必须在特定日期前执行完成，有的任务需要在另一个任务之后执行。

调度算法应该在满足特定的条件下，实现最大的下载量，提高提取速度和提取的吞吐量。

（4）网页自动结构化

1）对于电商类网页，能对同一个网站的数据进行自动结构化，生成不同的表，例如商品表、店铺表、评价表等

2）对于新闻博客类网页，能进行网页正文的自动抽取，对正文进行自动摘要和关键词分析

## 用户类和特征

（1）政府部门，拥有较多的数据库权限，可以将爬取的电商数据结合政府原有数据进行分析

（2）普通用户，拥有较少的权限，只能分析爬取的数据或者自行导入外部数据

（3）分析管理人员，拥有相应读写权限

## 运行环境

Python，运行在Python3.5

## 设计和实现上的限制

1. Python语言存在自身缺陷。
2. Web前端在不同浏览器兼容性不一致。

## 假设和依赖

（1）系统依赖于架设在云服务器的redis数据库。

（2）假设政府部门数据库接口请求得到批准。

（3）假设系统有足够的资源支持多用户同时请求。

## 关键点

关键的功能：网页自动结构化，反爬虫技术，通用性。

关键的算法：URL去重算法，分布式调度算法，HUSH算法。

# 外部接口需求

（1）政府相关部门数据库接口。

（2）用户希望自行导入的数据库的接口。

（3）爬虫子节点的ip。

（4）用户设置线程数量接口。

## 用户界面

系统运行于PC端，使用网页的方式呈现，遵循简约大气的风格，

登陆之后输入URL即可开始爬取，结果显示在如图2所示的表格中

## 硬件接口

无

## 软件接口

首先看一下个接口之间的关系图如图3.3-1所示：

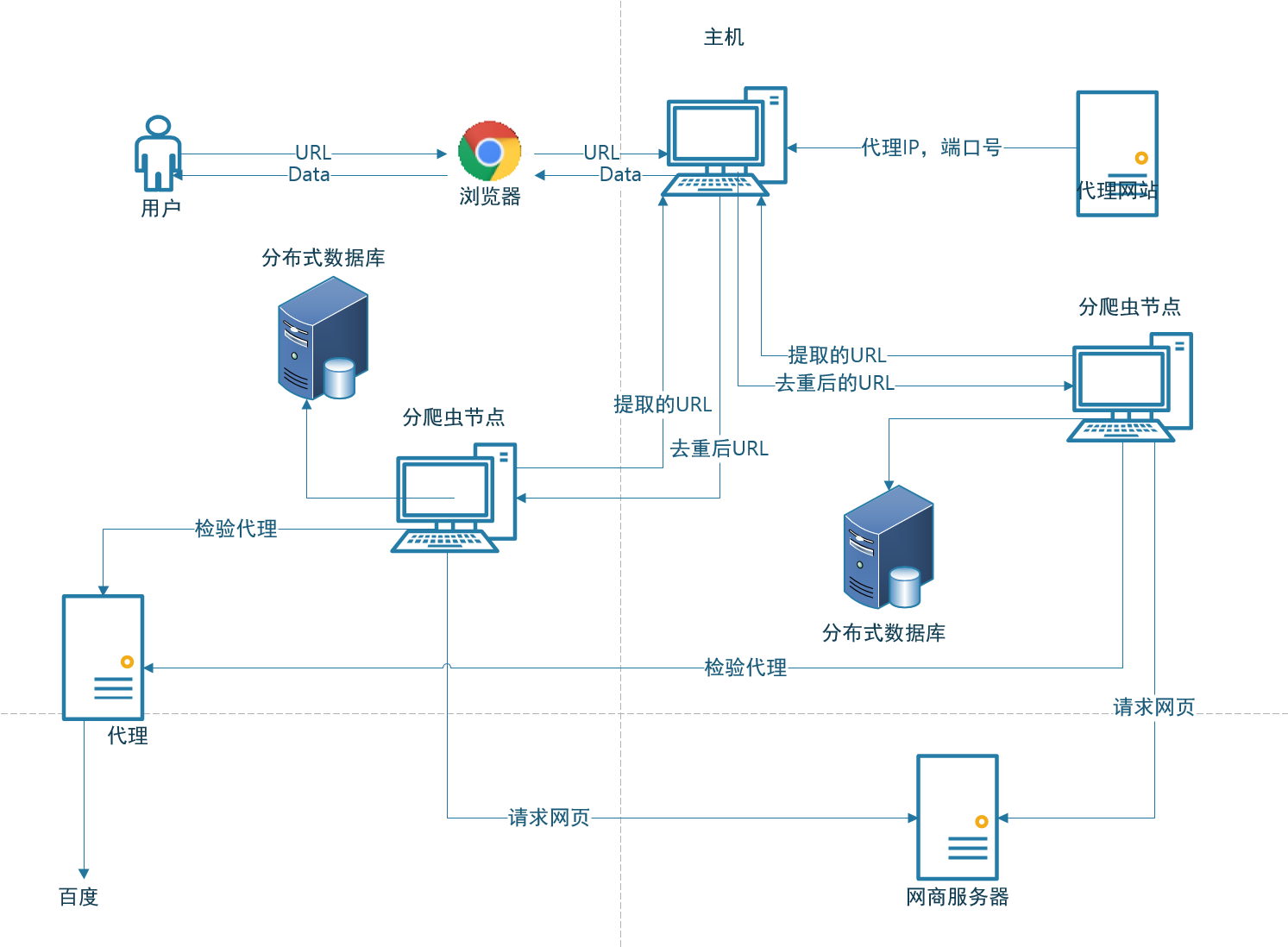


图3.3-1 各接口之间的关系

从图3.3-1可以看出，软件的接口和作用如下：

（1）用户输入和主机接口：用户输入想要爬取的URL，通过浏览器传给后端的主机

（2）主机和爬虫节点接口：主机将获得的URL分配给分爬虫节点，分爬虫节点将爬取的URL传给主机，主机去重后再将URL分配给各个分爬虫节点。

（3）爬虫节点和代理服务器的接口：获得代理ip和端口。

（4）爬虫节点和数据库的接口：将爬取的数据分布式的存储在数据中。

（5）爬虫节点和网商服务器接口：爬取电商网站得数据。

## 通信接口

（1）ip以及port的接口，从代理网站抓取与主机通信：将ip和port分发给爬虫子节点。

（2）从抓取的网页中提取的URL与爬虫子节点通信：将去重后的URL分发给爬虫子节点。

（3）前端web和tornado后端的通信：将后端爬取的数据和系统的性能数据传输给前端页面展示。

# 功能需求

## 功能分类

|  |  |
| --- | --- |
| **功能类别** | **功能** |
| 交互功能 | 与用户交互，输入URL，获得展示数据 |
| 主体功能 | 爬取 |
| 分析 |
| 请求/返回 |
| 管理功能 | 资源管理 |
| 存储管理 |
| 权限管理 |

## 系统用例图

系统用例图如图4.2-1所示。

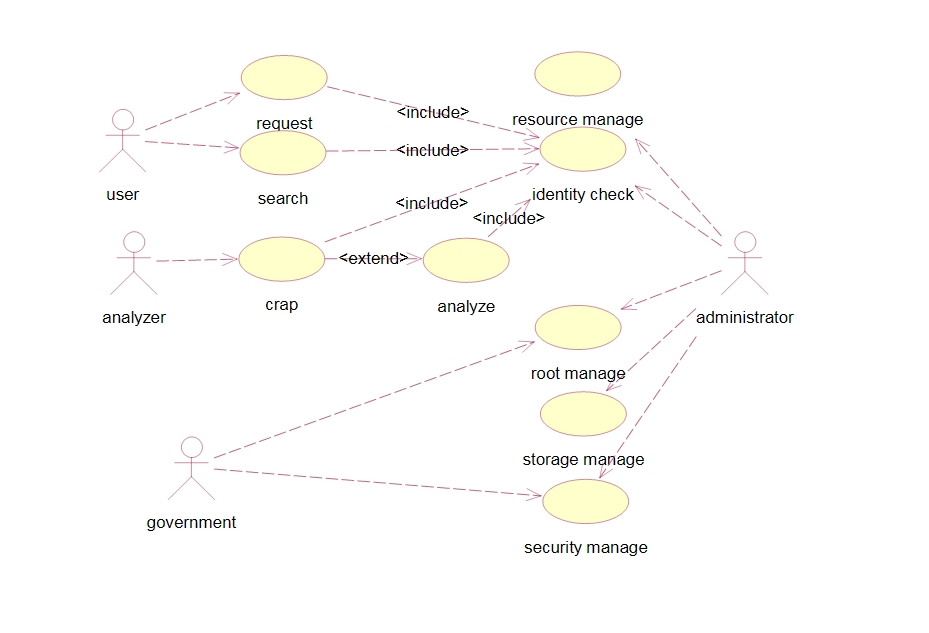


图4.2-1 系统用例图

## 功能模块图

总体得功能模块图和联系如图4.3-1所示。

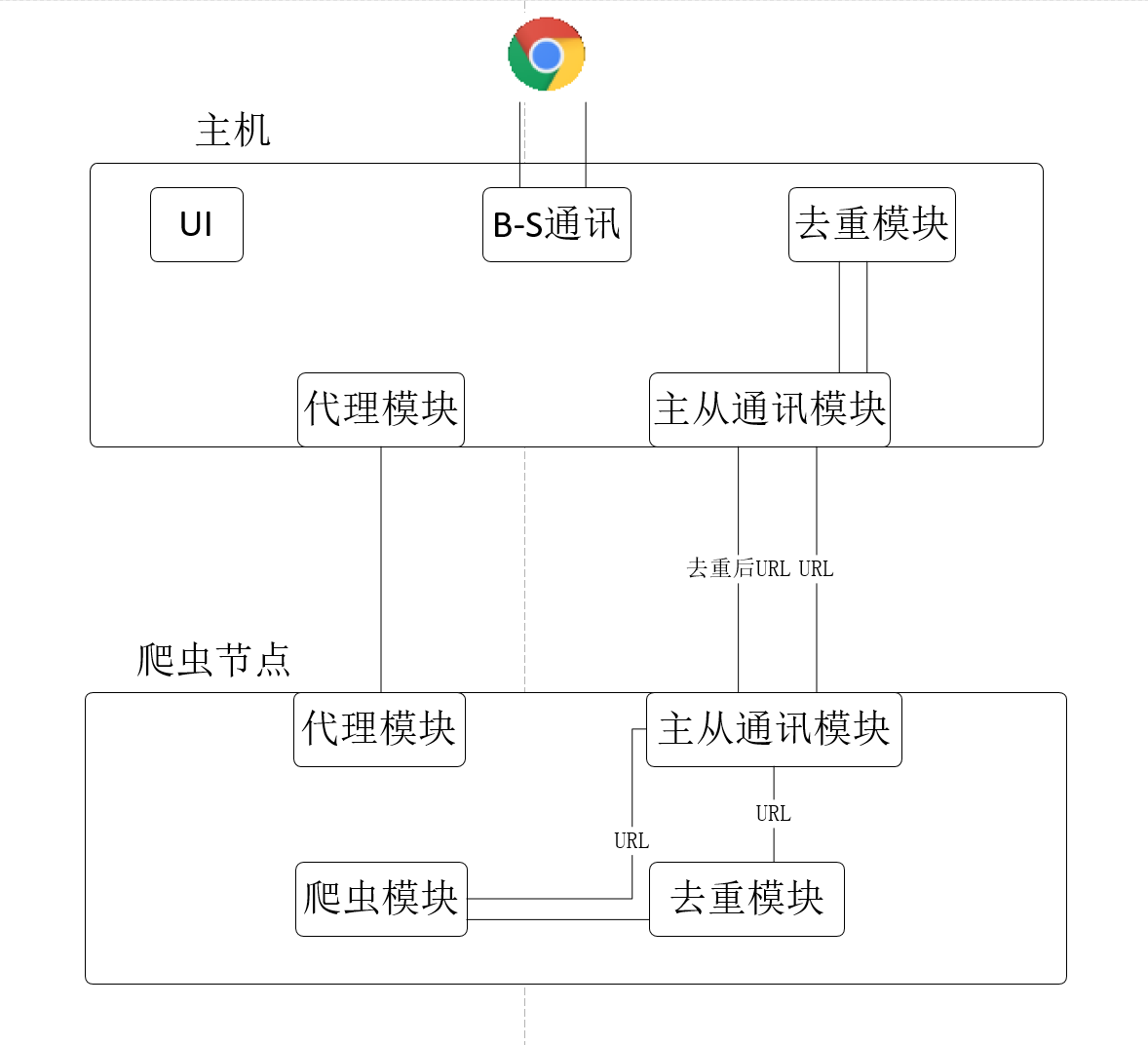


图4.3-1 功能模块图

一共有七个模块，分别如下：

1. 去重模块：对采集得URL进行去重。
2. 代理模块：获取代理IP和端口。
3. 爬虫主-从通讯模块：主机爬虫和节点爬虫进行通讯。
4. B-S通讯模块
5. 爬虫模块：爬取网页数据。
6. UI模块
7. 数据库模块：存储爬取数据。

## 程序流图

程序流图如图4.4-1所示。

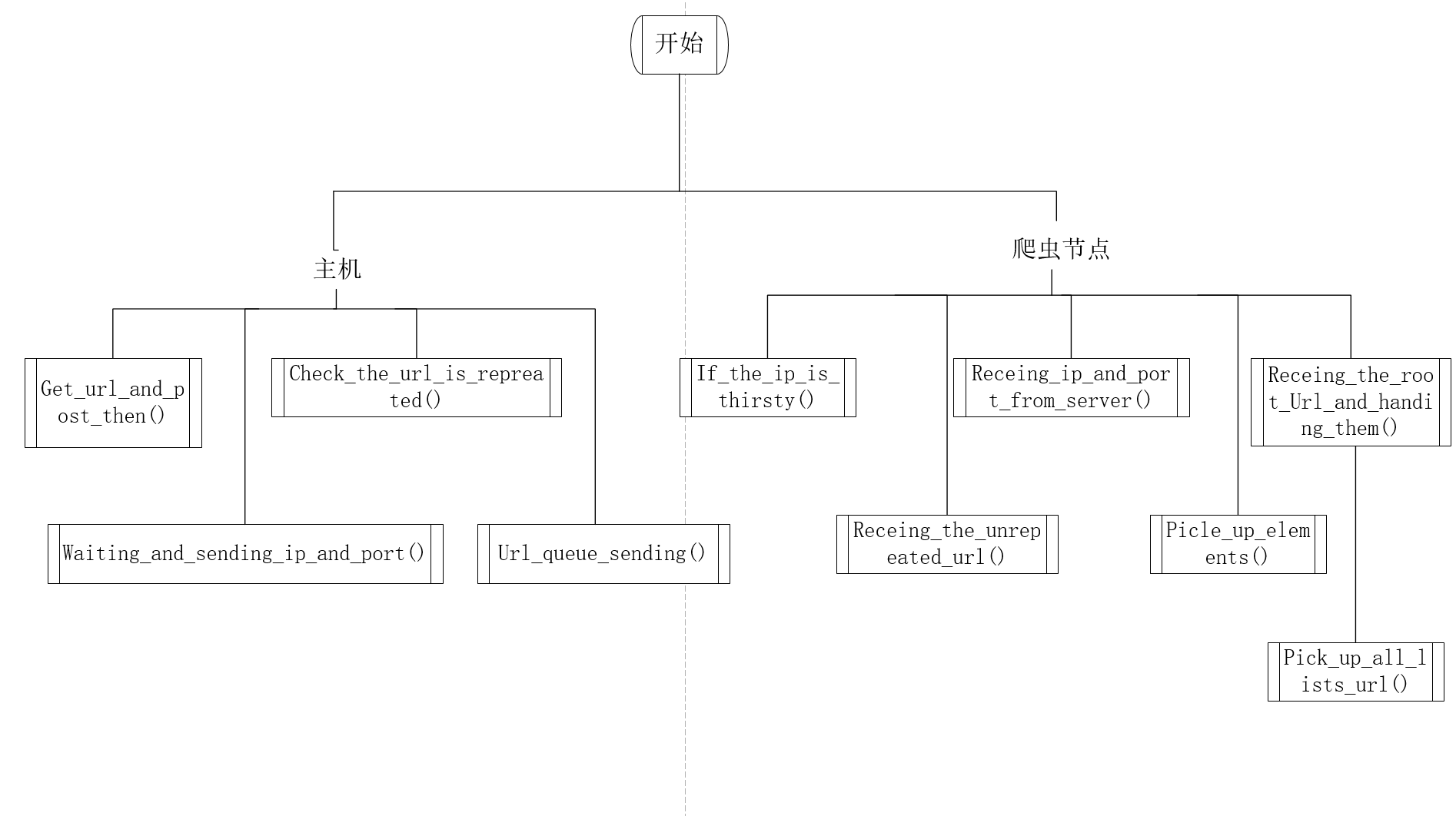


图4.4-1 程序流图

## 数据流图

（1）顶层数据流图如图4.5-1所示。

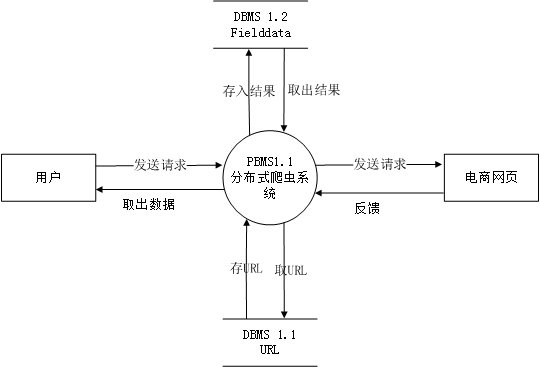


图4.5-1 顶层数据流图

（2）第一层数据流图如图4.5-2所示。

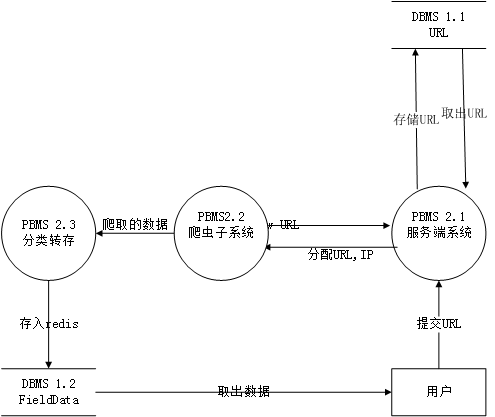


图4.5-2 第一层数据流图

（3）第二层数据流图如图4.5-3所示。

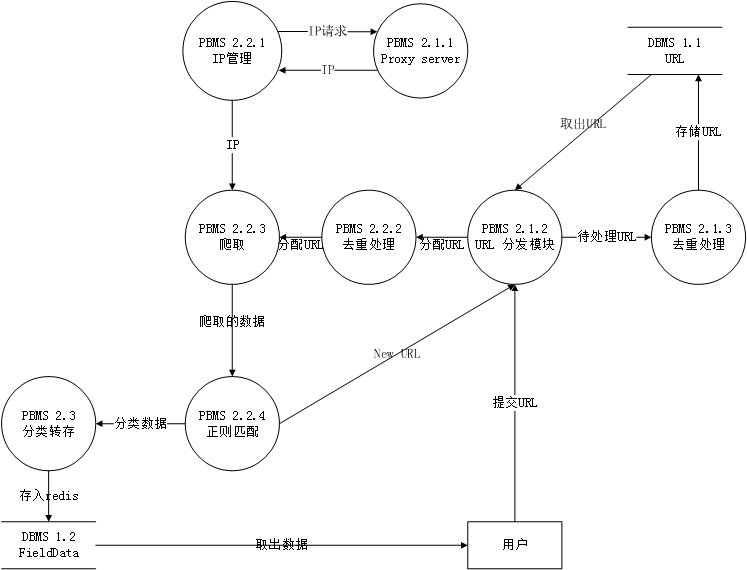


图4.5-3 第二层数据流图

# 非功能需求

## 性能需求

（1）在运行WINDOWS 7的2.6GHZ Intel i7 的计算机上，当系统至少有50%的空闲资源时，95%的目录数据库查询必须在两秒内完成。

（2）在拥有1台Windows2000的2G内存，1M带宽的云主机上，在网络条件顺畅的情况下，爬虫工作的时间必须可预知。

## 安全性需求

（1）每个用户在第一次登录之后，必须更改他的最初登录密码。最初的登录密码不能重用。

（2）系统出现意外时其数据可以使用备份还原，数据丢失不超过5%

（3）系统数据库必须拥有对病毒和黑客攻击的中级以上防御能力防止敏感数据泄露

## 软件质量属性

（1）软件没有入门技术门槛

（2）软件遵循C/B架构，可运行在任何平台的浏览器

## 其它需求

无

# 数据字典

## 实体定义

|  |  |  |  |
| --- | --- | --- | --- |
| **实体名称** | 店铺 | | |
| **实体描述** | 存储爬取的淘宝，京东或其他电商名下的店铺数据 | | |
| **属性名称** | **类型** | **精度** | **说明(属性的业务含义及业务规则)** |
| **店铺编号** | 字符 | 12 | 电商类别编号(3位)+店铺编号(9位) |
| **店铺名称** | 字符 | 20 | 店铺名称 |
| **所属类别** | 字符 | 5 | 店铺的分类 |
| **信誉** | 数字 | int | 信誉等级 |
| **评分** | 数字 | float | 4.0满分 |
| **商品数量** | 数字 | Int | 商品数量 |

|  |  |  |  |
| --- | --- | --- | --- |
| **实体名称** | 商品 | | |
| **实体描述** | 每个店铺的商品 | | |
| **属性名称** | **类型** | **精度** | **说明(属性的业务含义及业务规则)** |
| **商品编号** | 字符 | 6 | 商品的编号 |
| **店铺编号** | 字符 | 12 | 电商类别编号(3位)+店铺编号(9位) |
| **商品名称** | 字符 | 30 | 商品名称 |
| **商品价格** | 数字 | float | 价格 |
| **商品销量** | 数字 | int | 销量 |
| **好评比例** | 数字 | Float | 好评比例 |
| **商品类别** | 字符 | 20 | 商品类别 |
| **商品简介** | 字符 | 200 | 商品简介 |
| **商品货存** | 数字 | Int | 货存数量 |

|  |  |  |  |
| --- | --- | --- | --- |
| **实体名称** | 评论 | | |
| **实体描述** | 每个商品的评论 | | |
| **属性名称** | **类型** | **精度** | **说明(属性的业务含义及业务规则)** |
| **评论编号** | 字符 | 6 | 评论的编号 |
| **商品编号** | 字符 | 6 | 商品的编号 |
| **店铺编号** | 字符 | 12 | 电商类别编号(3位)+店铺编号(9位) |
| **评论内容** | 字符 | 200 | 评论内容 |

# 业务规则与业务算法

## 业务规则

（1）只有持有管理员密码的用户才能执行写操作。

（2）只有持有相应组权限的用户才能访问相应组的外部接口。

（3）对反复登陆失败，疑似非法，攻击性输入的情况，立刻查找输入IP并封锁

## 算法说明

分布式算法，就是指在完成乘加功能时通过将各输入数据每一对应位产生的运算结果预先进行相加形成相应的部分积，然后再对各部分进行累加形成最终结果。

分布式算法（Distributed Algorithm）和集中式算法（Centralized Algorithm）在设计的方法和技巧上，有着非常大的不同，原因在于分布式系统和集中式系统在系统模型和结构上有着本质的区别，集中式算法所具备的一些基本特征，在分布式算法中，已经不复存在。

分布式计算简单来说，是把一个大计算任务拆分成多个小计算任务分布到若干台机器上去计算，然后再进行结果汇总。 目的在于分析计算海量的数据，从雷达监测的海量历史信号中分析异常信号(外星文明)，淘宝双十一实时计算各地区的消费习惯等。

海量计算最开始的方案是提高单机计算性能，如大型机，后来由于数据的爆发式增长、单机性能却跟不上，才有分布式计算这种妥协方案。 因为计算一旦拆分，问题会变得非常复杂，像一致性、数据完整、通信、容灾、任务调度等问题也都来了。

举个例子，产品要求从数据库中100G的用户购买数据，分析出各地域的消费习惯金额等。 如果没什么时间要求，程序员小明就写个对应的业务处理服务程序，部署到服务器上，让它慢慢跑就是了，小明预计10个小时能处理完。 后面产品嫌太慢，让小明想办法加快到3个小时。

　　平常开发中类似的需求也很多，总结出来就是，数据量大、单机计算慢。 如果上Hadoop、storm之类成本较高、而且有点大才小用。 当然让老板买更好的服务器配置也是一种办法。

分布性和并发性是分布式算法的两个最基本的特征。分布式系统的执行存在着许多非稳定性的因素。由于这些多方面的差异，导致分布式算法的设计和分析，较之集中式算法来讲，要复杂得多，也困难得多。

所谓分布式算法，就是指在完成乘加功能时通过将各输入数据每一对应位产生的运算结果预先进行相加形成相应的部分积，然后再对各部分进行累加形成最终结果。

而传统算法则是等到所有乘积结果之后再进行相加，从而完成整个乘加运算。

在我们的分布式爬虫当中，存在一个主机，负责集中去重（将各个爬虫节点提取的URL集中进行去重），然后平均分配给各个爬虫分节点，爬虫分节点在将提取的URL传给主机前会先去重过滤一次。

我们的主机的去重用的是python自带的容器去重。

# 附录A：分析模型

具体在第四章中有分析，这里是主要的代码分析

C:.  
│  README  
├─main\_server  
│  │  ghostdriver.log  
│  │  IMG_256lock.py  
│  │  phantomjs.exe  
│  │  IMG_257sender.py  
│  │  server\_IMG_258main.py  
│  │  \_\_init\_\_.py  
│  │    
│  ├─de\_weight  
│  │  │  de\_IMG_259weight.py  
│  │  │  \_\_init\_\_.py  
│  │  │    
│  │  └─\_\_pycache\_\_  
│  │          de\_weight.cpython-35.pyc  
│  │          \_\_init\_\_.cpython-35.pyc  
│  │            
│  ├─server\_ip\_and\_port  
│  │  │  IMG_260lock.py  
│  │  │  post\_get\_IMG_261proxy.py  
│  │  │  sending\_ip\_and\_IMG_262url.py  
│  │  │  TCP\_IMG_263receiver.py  
│  │  │  \_\_init\_\_.py  
│  │  │    
│  │  └─\_\_pycache\_\_  
│  │          lock.cpython-35.pyc  
│  │          post\_get\_proxy.cpython-35.pyc  
│  │          sending\_ip\_and\_url.cpython-35.pyc  
│  │          TCP\_receiver.cpython-35.pyc  
│  │          \_\_init\_\_.cpython-35.pyc  
│  │            
│  └─\_\_pycache\_\_  
│          lock.cpython-35.pyc  
│          sender.cpython-35.pyc  
│          server\_main.cpython-35.pyc  
│          \_\_init\_\_.cpython-35.pyc  
│            
├─redis  
│      IMG_264redis-store.py  
│        
└─spider  
    │  create\_IMG_265driver.py  
    │  ghostdriver.log  
    │  IMG_266lock.py  
    │  phantomjs.exe  
    │  IMG_267receiver.py  
    │  IMG_268sender.py  
    │  spider\_IMG_269main.py  
    │  webdriver\_change\_IMG_270proxy.py  
    │    
    ├─de\_weight  
    │  │  de\_IMG_271weight.py  
    │  │  \_\_init\_\_.py  
    │  │    
    │  └─\_\_pycache\_\_  
    │          de\_weight.cpython-35.pyc  
    │          \_\_init\_\_.cpython-35.pyc  
    │            
    ├─spider\_ip\_and\_port  
    │  │  check\_IMG_272proxy.py  
    │  │  IMG_273lock.py  
    │  │  IMG_274receiver.py  
    │  │  receiving\_ip\_and\_IMG_275port.py  
    │  │  TCP\_IMG_276sender.py  
    │  │  \_\_init\_\_.py  
    │  │    
    │  └─\_\_pycache\_\_  
    │          check\_proxy.cpython-35.pyc  
    │          receiving\_ip\_and\_port.cpython-35.pyc  
    │          TCP\_sender.cpython-35.pyc  
    │          \_\_init\_\_.cpython-35.pyc  
    │            
    └─\_\_pycache\_\_  
            create\_driver.cpython-35.pyc  
            lock.cpython-35.pyc  
            receiver.cpython-35.pyc  
            webdriver\_change\_proxy.cpython-35.pyc

# 附 录B：待确定问题的列表

（1）如何区别对待不同类型的用户，划分数据的权限尚未确定。

（2）没有对系统进行压力测试，性能测试，安全测试，其风险未知。

# 附录C：编写文档的原则

* 使用的术语与词汇表中所定义的术语一致。
* 避免模糊的、主观的术语如用户友好、容易、简单、迅速、有效、许多、最新技术、优越的、可接受的、健壮的等等。
* 避免使用比较性的词汇如提高、最大化、最小化、最佳化等。定量说明所需要提高的程度或者说清一些参数可以接受的最大值和最小值。含糊的语句表达将引起需求的不可验证。
* 把顶层不明确的需求向低层详细分解，直到消除不明确性为止。
* 编写单个的可测试需求文档。
* 不应该把多个需求集中在一个冗长的叙述段落中，不在需求说明中使用“和/或”，“等等”之类的连词。