**分布式爬虫系统**



**电子科技大学**

**iMake**

小组成员：郑毅，王骞，祝含颀 指导教师：吴晓华

[分布式爬虫系统需求规格说明书 1](#_Toc6964)

[分布式爬虫系统系统设计 2](#_Toc17181)1

[分布式爬虫系统项目管理 2](#_Toc6070)9

[分布式爬虫系统使用说明书 3](#_Toc13758)4

[分布式爬虫系统总结 3](#_Toc29837)7

[分布式爬虫系统测试文档 4](#_Toc6867)0

[分布式爬虫系统作品简介与参赛感言 5](#_Toc16812)3

**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：

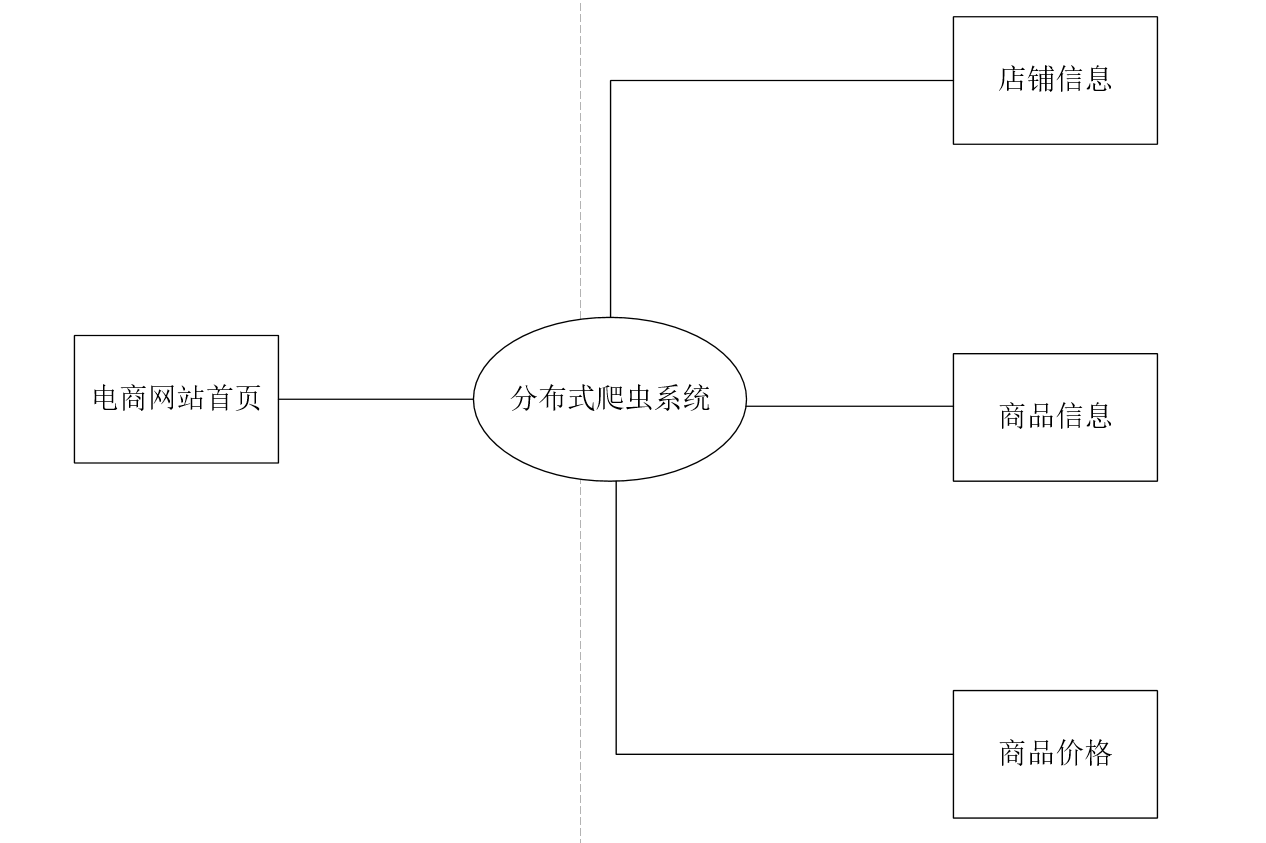


图2.2-1 分布式爬虫顶层数据流图

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：

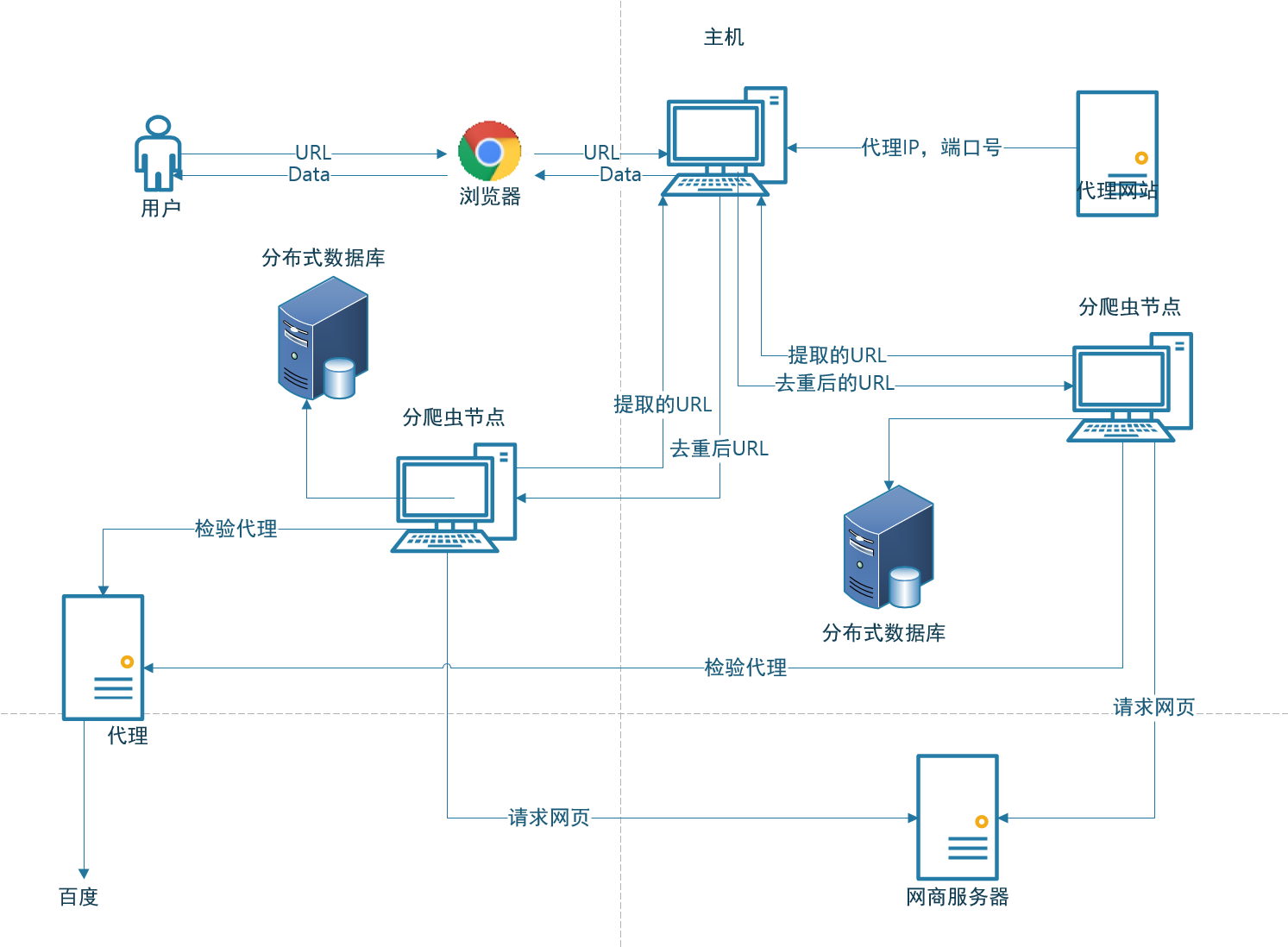


图3.3-1 各接口之间的关系

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。

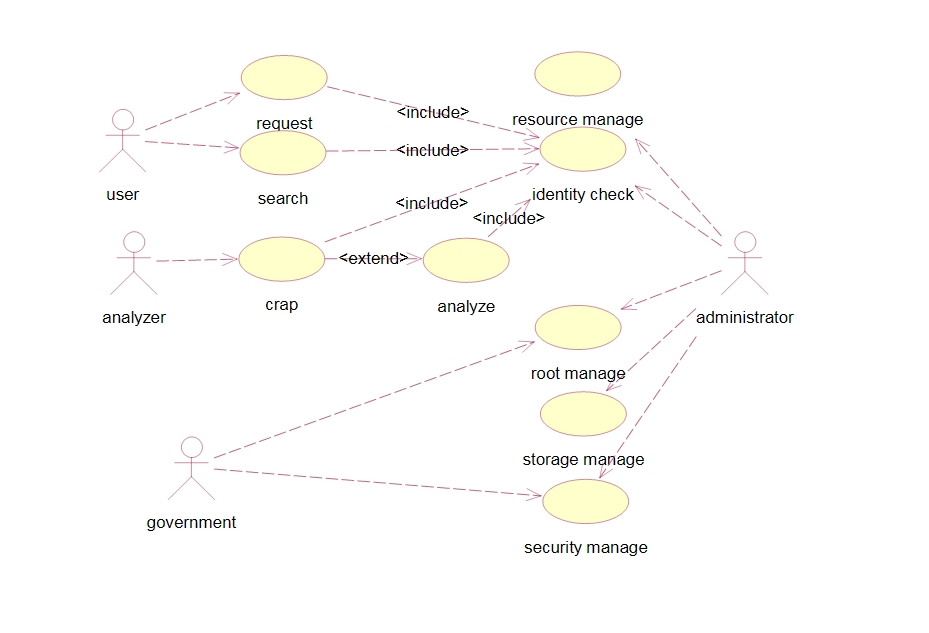


图4.2-1 系统用例图

## 功能模块图

总体得功能模块图和联系如图4.3-1所示。

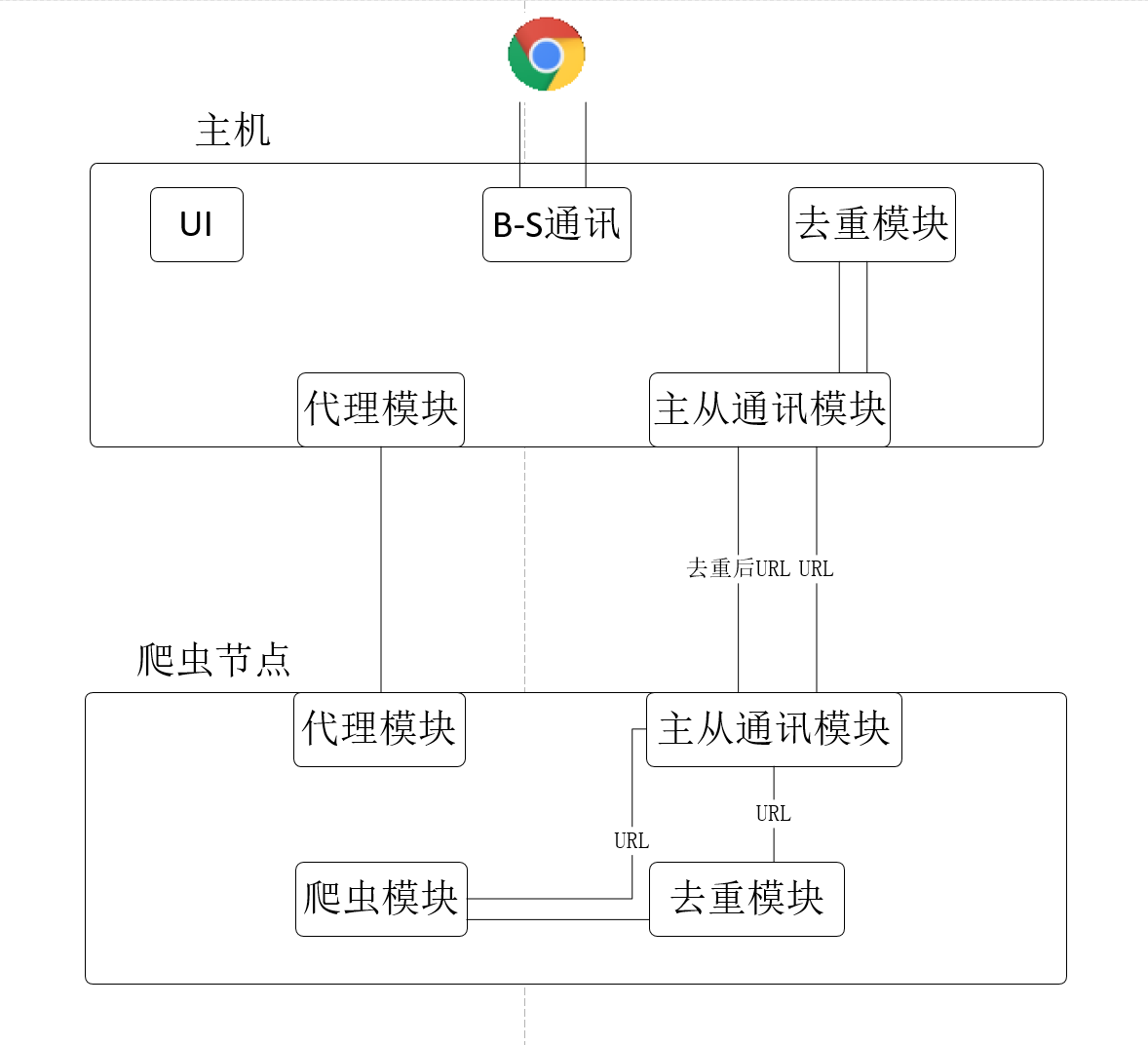


图4.3-1 功能模块图

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。

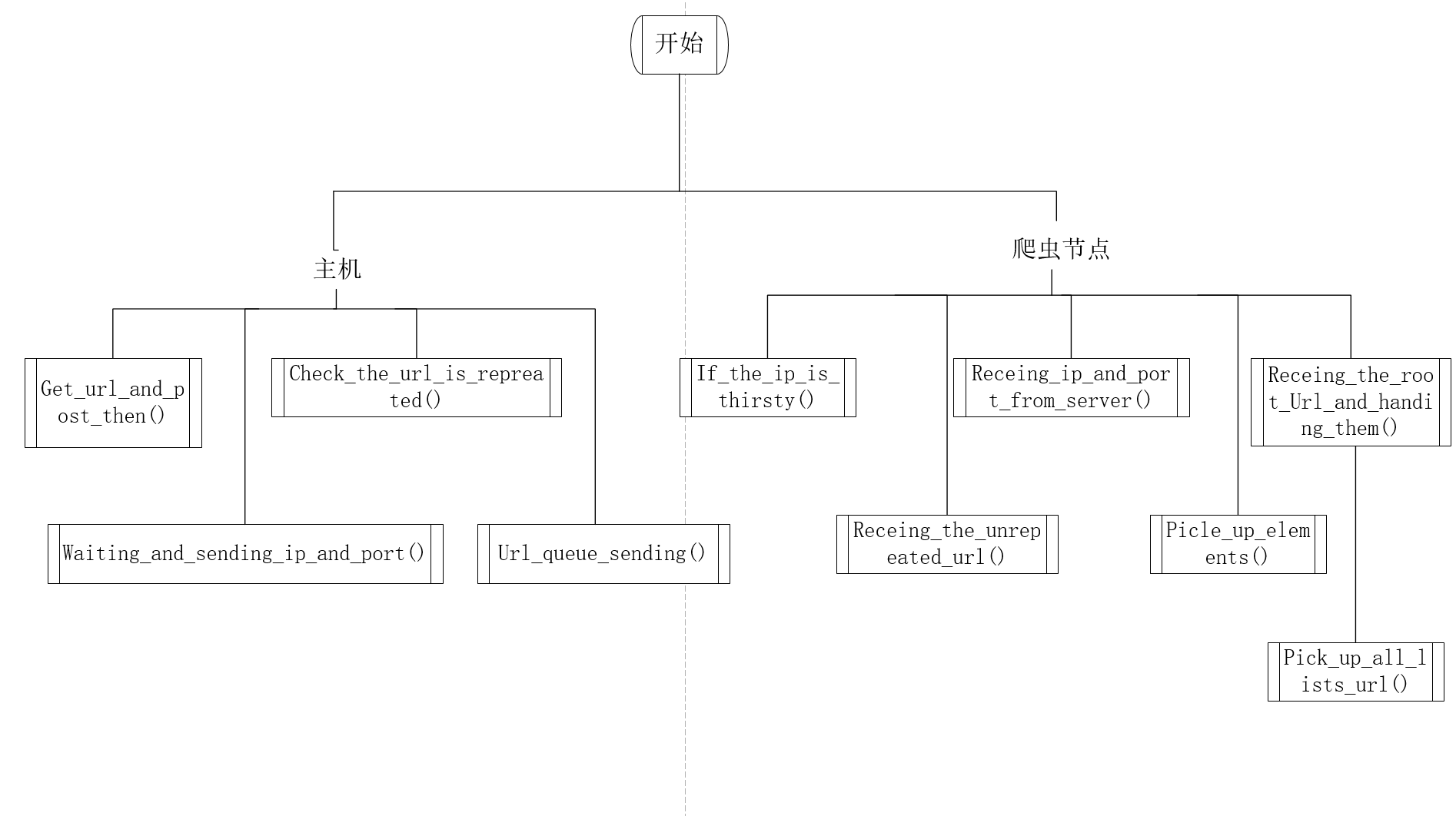


图4.4-1 程序流图

## 数据流图

（1）顶层数据流图如图4.5-1所示。

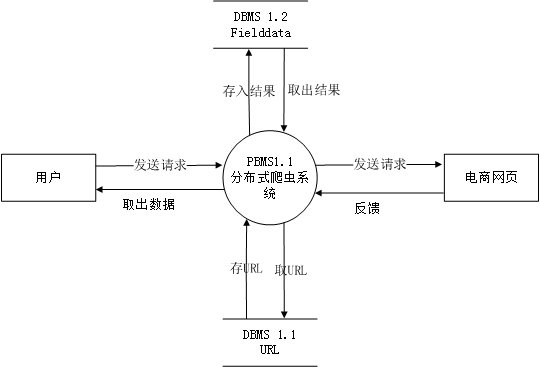


图4.5-1 顶层数据流图

（2）第一层数据流图如图4.5-2所示。

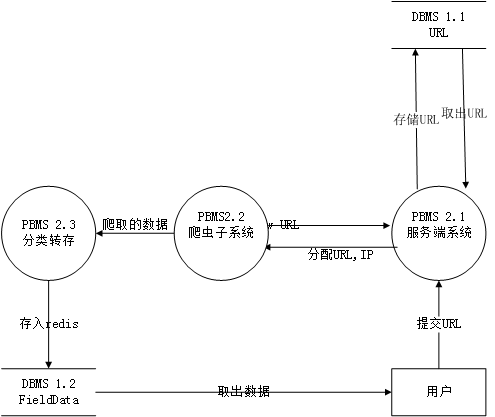


图4.5-2 第一层数据流图

（3）第二层数据流图如图4.5-3所示。

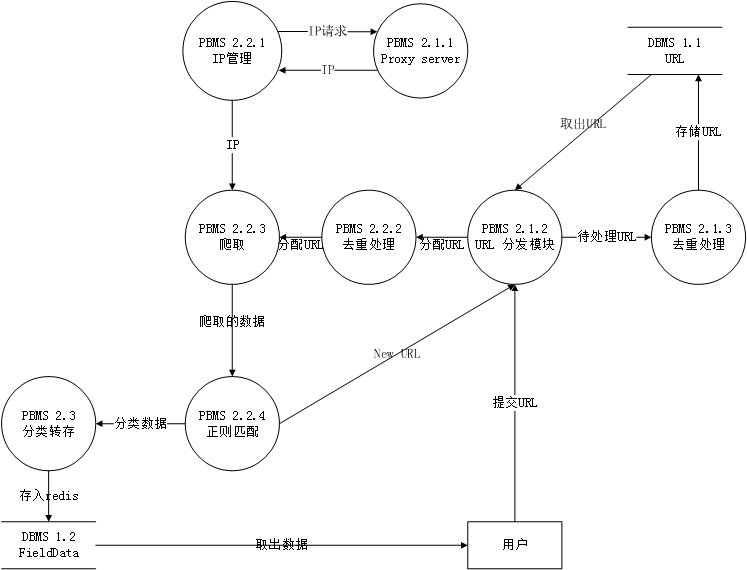


图4.5-3 第二层数据流图

# 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。

**分布式爬虫系统系统设计**



1. **系统功能模块**

设计分布式爬虫架构，架构满足以下功能

1. 分布式：爬虫应该能够在多台机器上分布执行。
2. 可伸缩性：爬虫结构应该能够通过增加额外的机器和带宽来提高抓取速度。
3. 性能和有效性：爬虫系统必须有效地使用各种系统资源，例如，处理器、存储空间和网络带宽。
4. 去重能力：分爬虫所抓取的URL数量众多，会出现重复的情况，应当集中的处理分发。

设计的功能模块的框架图如图1-1所示。

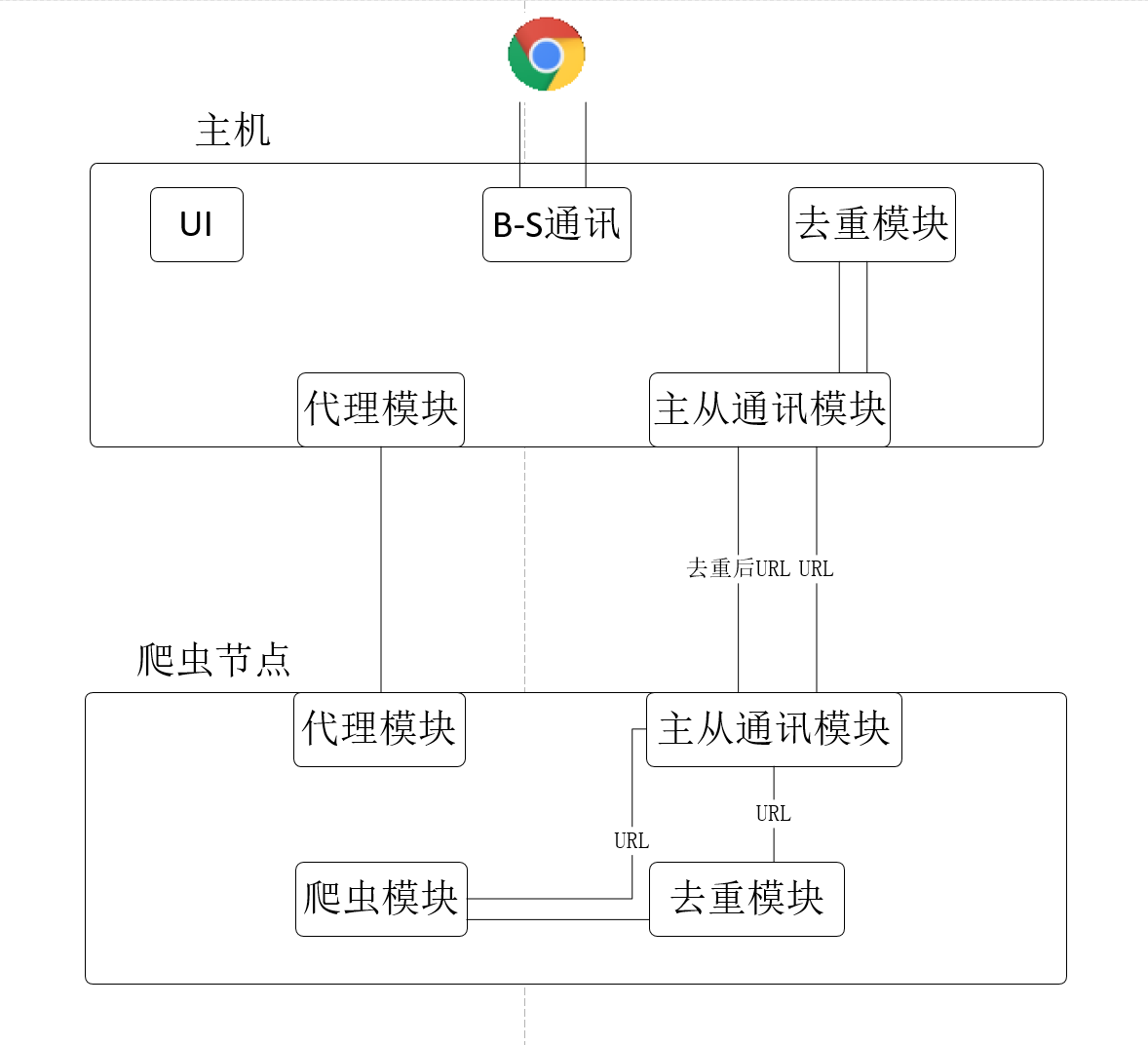


图 1-1 功能模块图

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。

R-B Tree，全称是Red-Black Tree，又称为“红黑树”，它一种特殊的二叉查找树。红黑树的每个节点上都有存储位表示节点的颜色，可以是红(Red)或黑(Black)。

红黑树的特性:

（1）每个节点或者是黑色，或者是红色。

（2）根节点是黑色。

（3）每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]

（4）如果一个节点是红色的，则它的子节点必须是黑色的。

（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。

注意：

(01) 特性(3)中的叶子节点，是只为空(NIL或null)的节点。

(02) 特性(5)，确保没有一条路径会比其他路径长出俩倍。因而，红黑树是相对是接近平衡的二叉树。

红黑树示意图如图1-2所示：

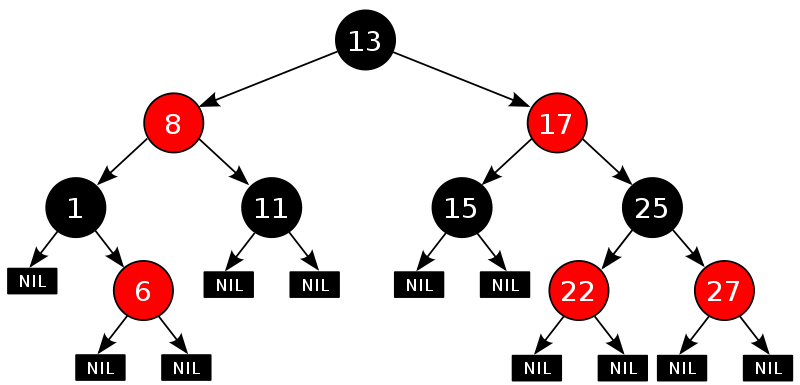


图 1-2 红黑树示意图

红黑树的应用：

红黑树的应用比较广泛，主要是用它来存储有序的数据，它的时间复杂度是O(lgn)，效率非常之高。

例如，Java中的TreeSet和TreeMap，C++ STL中的set、map，以及Linux虚拟内存的管理，都是通过红黑树去实现的。

这里大致介绍下，红黑树和AVL树的差异。AVL树也是特殊的二叉树，它的特性是“任何节点的左右子树的高度之差不超过1”。基本上，用到红黑树的地方都可以用AVL树(自平衡二叉查找树)去替换。但是一般情况下，在执行添加、删除节点时，AVL树比红黑树执行的操作更多一些，效率更低一些；而且红黑树也是相对平衡的二叉树(从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点)。因此，红黑树的效率会高更一点。

R-B Tree时间复杂度

红黑树的时间复杂度为: O(lgn)

下面通过“数学归纳法”对红黑树的时间复杂度进行证明。

定理：一棵含有n个节点的红黑树的高度至多为2log(n+1).

证明：

"一棵含有n个节点的红黑树的高度至多为2log(n+1)" 的逆否命题是 "高度为h的红黑树，它的包含的内节点个数至少为 2^{h/2}-1个"。

我们只需要证明逆否命题，即可证明原命题为真；即只需证明 "高度为h的红黑树，它的包含的内节点个数至少为 2^{h/2}-1个"。

从某个节点x出发（不包括该节点）到达一个叶节点的任意一条路径上，黑色节点的个数称为该节点的黑高度，记为bh(x)。

由红黑树的"特性(4)"可知 bh(x)>=h/2；进而，我们只需证明 "高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"即可。

到这里，我们将需要证明的定理已经由

"一棵含有n个节点的红黑树的高度至多为2log(n+1)"

转变成只需要证明

"高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"。

下面通过"数学归纳法"开始论证高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"。

(1) 当树的高度h=0时，

内节点个数是0，bh(x) 为0，2^bh(x)-1 也为 0。显然，原命题成立。

(2) 当h>0，且树的高度为 h-1 时，它包含的节点个数至少为 2^{bh(x)-1}-1。这个是根据(01)推断出来的！

下面，由树的高度为 h-1 的已知条件推出“树的高度为 h 时，它所包含的节点树为 2^bh(x)-1”。

当树的高度为 h 时，

对于节点x(x为根节点)，其黑高度为bh(x)。

对于节点x的左右子树，它们黑高度为 bh(x) 或者 bh(x)-1。

根据(02)的已知条件，我们已知 "x的左右子树，即高度为 h-1 的节点，它包含的节点至少为 2^{bh(x)-1}-1 个"；

所以，节点x所包含的节点至少为 ( 2^{bh(x)-1}-1 ) + ( 2^{bh(x)-1}-1 ) + 1 = 2^{bh(x)-1}。即节点x所包含的节点至少为 2^{bh(x)-1} 。

因此，原命题成立。

由(1)、(2)得出，"高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"。

因此，“一棵含有n个节点的红黑树的高度至多为2log(n+1)”。

1. 代理模块：获取代理IP和端口。

开启线程检测代理堆栈是否为空，若为空则发送请求索要新的代理IP（期间设有信号量阻塞请求）。代理的检验环节交给爬虫分节点以避免中心节点负担过重。

1. 爬虫主-从通讯模块：主机爬虫和节点爬虫进行通讯。

主机负责集中去重，任务调配还有代理的抓取还有分发。从机负责验证代理可用性，爬取信息，采集数据，分布式存储，回传待去重的URL。

1. B-S通讯模块

B-S通讯模块基于websocket的通讯，介绍一下websocket。

WebSocket是HTML5出的协议，也就是说HTTP协议没有变化，但HTTP是不支持持久连接的（长连接，循环连接的不算）。  
 首先HTTP有1.1和1.0之说，也就是所谓的keep-alive，把多个HTTP请求合并为一个，但是Websocket其实是一个新协议，跟HTTP协议基本没有关系，只是为了兼容现有浏览器的握手规范而已，也就是说它是HTTP协议上的一种补充，可以通过图1-3理解他们之间的关系。

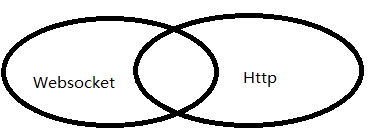


图1-3 websocket和http的关系

Websocket是一个持久化的协议，相对于HTTP这种非持久的协议来说，websocket的主要是为了解决在web上应用长连接进行灵活的通讯应用而产生，但websocket本身只是一个基础协议，对于消息上还不算灵活，毕竟websocket只提供文本和二进制流这种基础数据格式．在实际应用则更偏向于对象消息的处理，而在这基础上更希望集成一系列的消息路由机制来解决消息处理上的问题．为了解决以上问题beetle针对websocket进行了一些高层次的封装，让服务端处理消息变得更简单灵活。

1. 爬虫模块：爬取网页数据。

使用面向对象方法封装PhantomJS的一系列操作，请求的方法中附带有触发器，在请求次数达到上限的时候自动进行IP的切换。使用正则表达式引擎模糊提取信息，存放在数据库中。

1. UI模块

UI模块用基于websocket通信的Echarts的动态数据展示柱状图和折线图，时刻向用户展示爬取数据的信息和爬虫系统的性能。ECharts，一个纯 Javascript 的图表库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），底层依赖轻量级的 Canvas 类库 ZRender，提供直观，生动，可交互，可高度个性化定制的数据可视化图表。ECharts 提供了常规的折线图，柱状图，散点图，饼图，K线图，用于统计的盒形图，用于地理数据可视化的地图，热力图，线图，用于关系数据可视化的关系图，treemap，多维数据可视化的平行坐标，还有用于 BI 的漏斗图，仪表盘，并且支持图与图之间的混搭。

1. 数据库模块：存储爬取数据。

数据库采用redis，redis数据库具有存储数据快，多线程支持好，可移植性强等特点。

**2.程序流程图**

整体的程序流程图如图2-1所示。

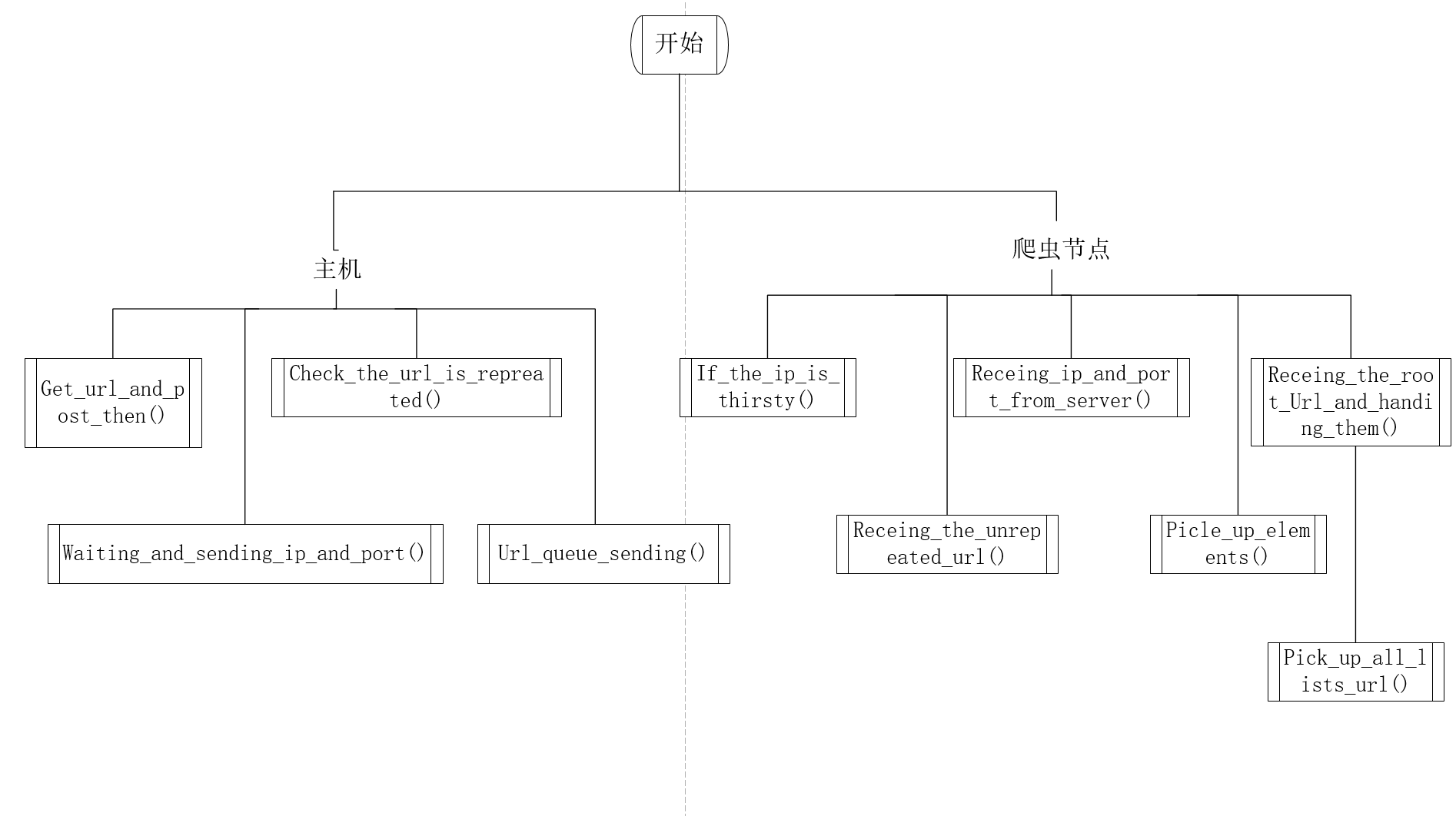


图 2-1 程序流图

整体的流程思路是：用户从浏览器界面输入待提取的网页，如www.taobao.com.浏览器将该网页传给爬虫主机，爬虫主机获得最高域名的网页www.taobao.com后开始爬取出主页面下的各个URL，主机将这一批URL平均分发给各个爬虫分节点爬取，各个爬虫分节点获取URL，自己过滤后再传给主机爬虫，主机爬虫将这些URL记录去重后再分发给爬虫分节点爬取，各个分节点爬取的数据储存在各个分节点的数据库中。

其中爬虫模块的流程图如图2-2所示。



图 2-2 爬虫模块流程图

爬虫模块的时序图如图2-3所示。

数据库

结构化

爬虫节点

分配

初始化URL

去重

1.提交

2.分配

3.爬取

4.存储

5.提交URL

6.去重

7.分配

8.爬取

9.存储

图 2-3 爬虫模块时序图

总体的框架接口有：

（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。

总体的接口图如图2-4所示。

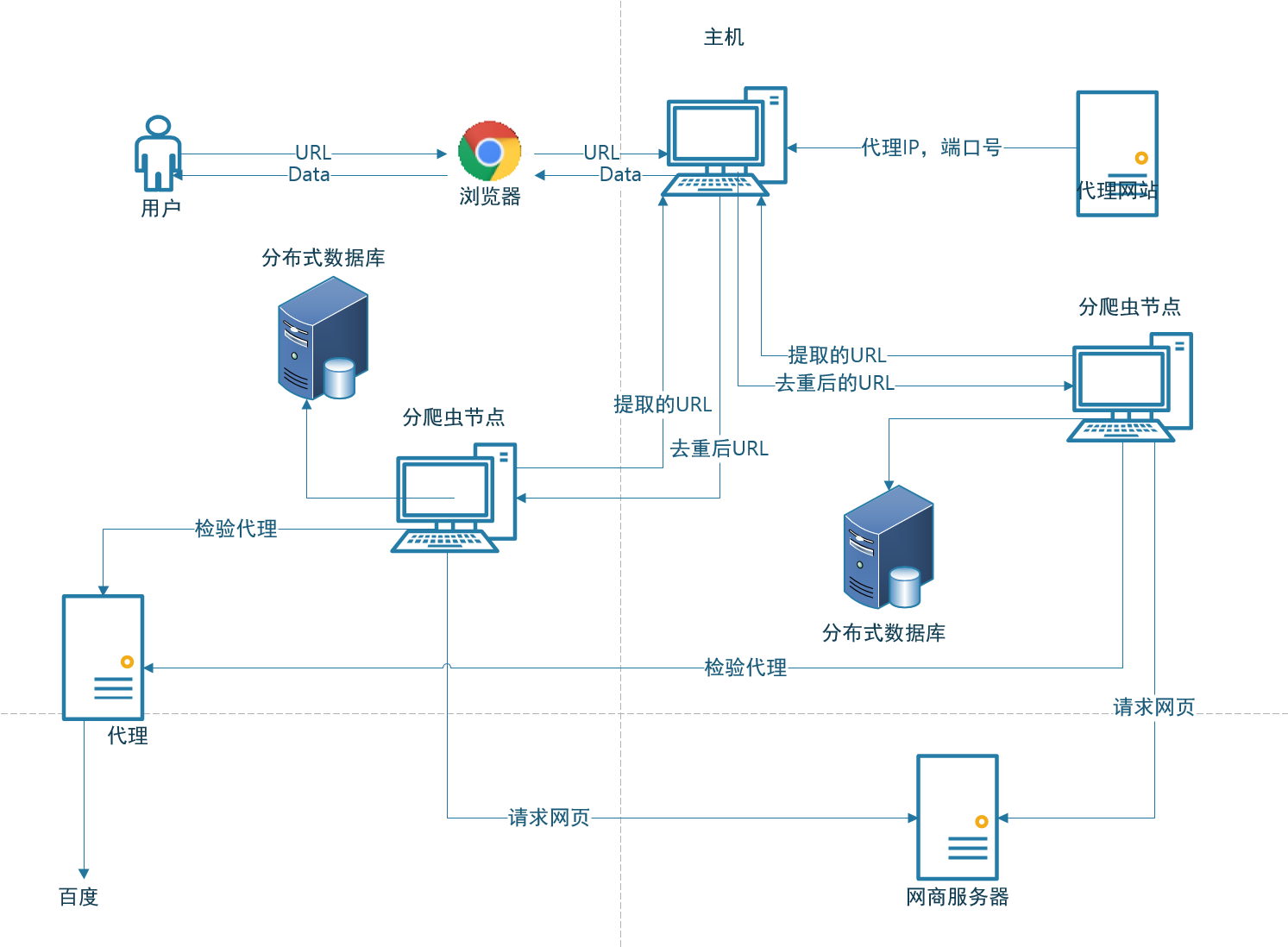


图2-4 总体框架接口图

**分布式爬虫系统项目管理**



**1.绘制项目的活动网络图**

分布式爬虫系统的活动网络图如图1.1所示。

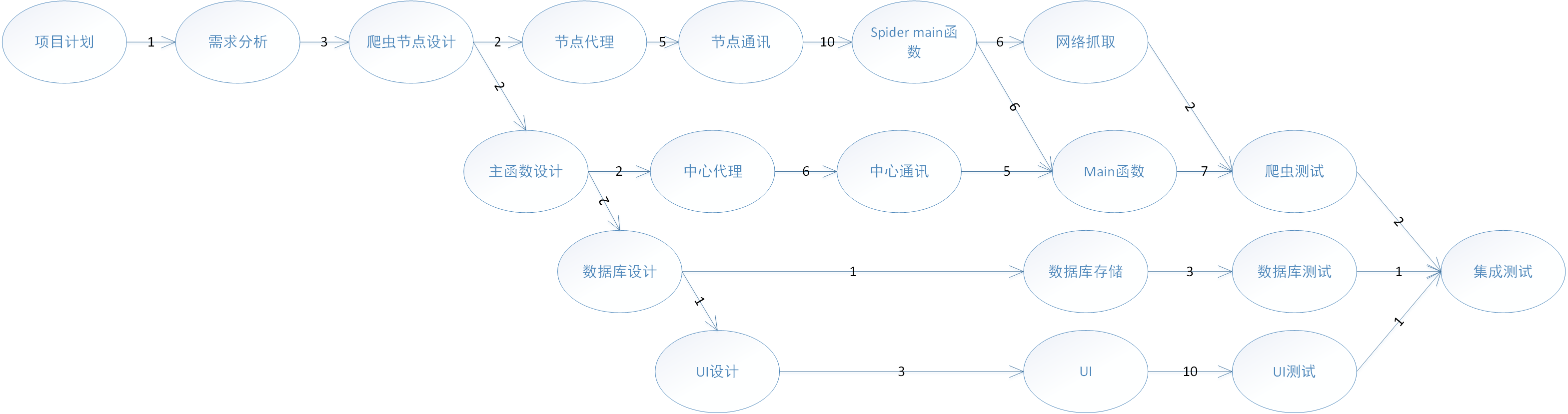


图1.1 活动网络图

图1.1完整的描述了项目设计与执行的全部流程，以项目计划为开始节点，集成测试为结束节点，分为爬虫节点、主函数、数据库和UI四个部分分别执行。每部分大体由设计、实现、测试三个小模块组成，最后经过集成测试完成对项目的总体测试。每一个活动预期需要耗费的时间都作出了标注。

**2.利用功能点法度量软件（活动）的工作量**

功能点计算方法 ：

FP = UFC×TCF = UFC × (0.65 + 0.01×Fi)

UFC (Unadjusted Function Component) : 未调整功能点计数 5个信息量的“加权和” 功能点计数, 5个信息量的 加权和

TCF (Technical Complexity Factor): 技术复杂度因子

Fi: 14个因素的“复杂性调节值” (i =1..14)

0.65, 0.01：经验常数

UFC：

|  |  |  |  |
| --- | --- | --- | --- |
| 输入文件 | 2 | URL  分析方向 | 2\*3=6 |
| 输出文件 | 3 | 分析表  聚类分析图  参数预测图 | 1\*4+2\*5=14 |
| 查询 | 1 | 查询数据库 | 1\*3=3 |
| 内部文件 | 2 | 网站信息  已爬取URL表 | 2\*7=14 |
| 外部文件 | 2 | 安全信息  政府监管信息 | 1\*7+1\*10=17 |

表2.1 UFC预测表

由表2.1可知：UFC=6+14+3+14+17=54

TCP：

|  |  |
| --- | --- |
| 系统特征 | 复杂度 |
| 数据通讯 | 2 |
| 数据分布处理 | 4 |
| 系统性能 | 5 |
| 使用配置 | 2 |
| 事务处理频率 | 2 |
| 在线数据输入 | 4 |
| 操作便利 | 1 |
| 在线更新 | 1 |
| 处理复杂度 | 2 |
| 复用性 | 2 |
| 安装难易 | 1 |
| 运行维护 | 1 |
| 多站点支持 | 1 |
| 变更支持 | 4 |

表2.2 复杂度调整表

由表2.2可知：

TCF=0.65+0.01\*（1\*5+3\*4+5\*2+5\*1）=0.87

经过复杂度调整后，可计算出调整后的功能点：

FP=UFC\*TCF=46.98

1. **利用制定项目的进度计划**

根据项目活动网络图及经复杂度调整后计算出的项目功能点，设计出图3.1所示的项目进度计划，项目从2017年4月27日开始，到6月26日完成所有测试并进入为期60个工作日的维护阶段，预计在9月28日完成整个项目。

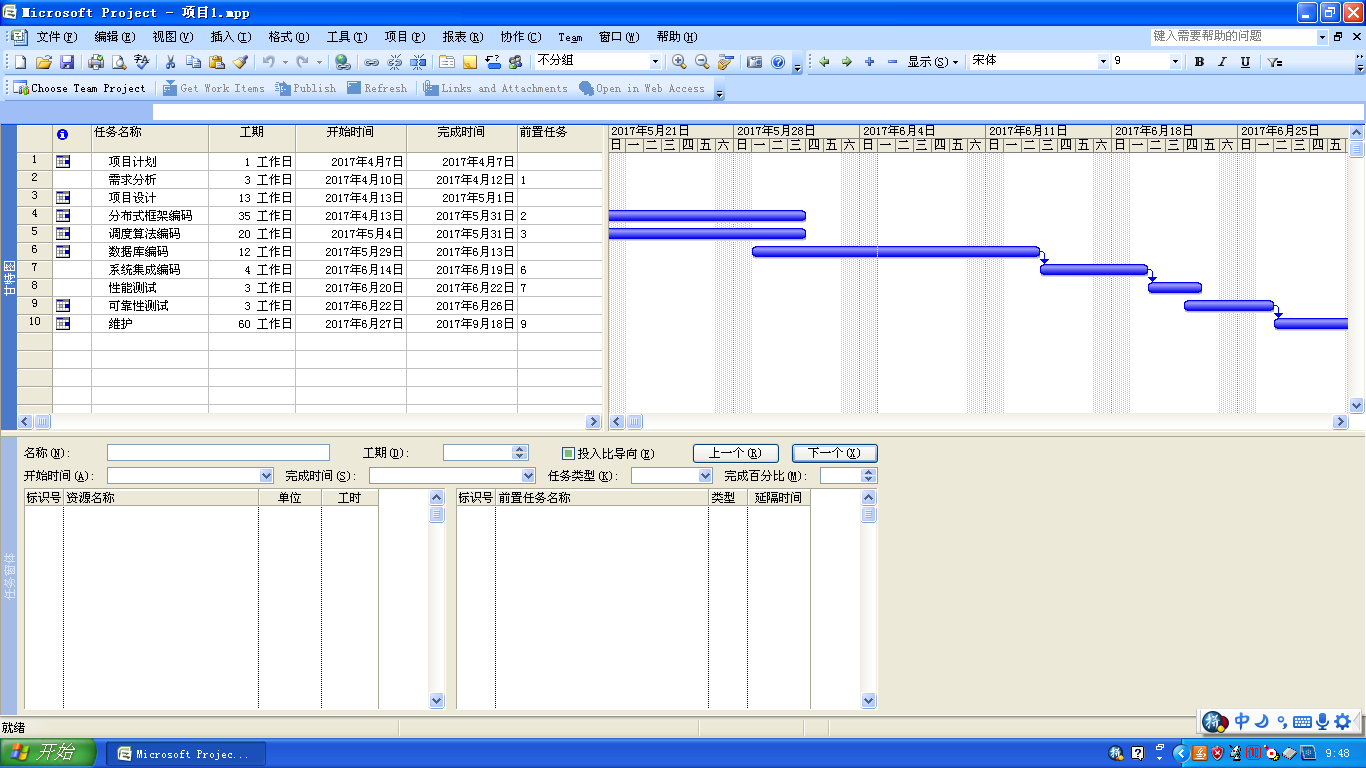


图3.1 项目进度计划

1. **计算关键路径**

根据项目进度计划，绘制了图4.1所示的关键路径图，为每个节点分配了各自完成所需的最大工作日期限，通过CPM关键路径算法得出了本项目的关键路径，即项目计划、需求分析、爬虫节点设计、结点代理、节点通讯、Spider main函数、Main函数、爬虫测试、集成测试，关键路径长度为36工作日。

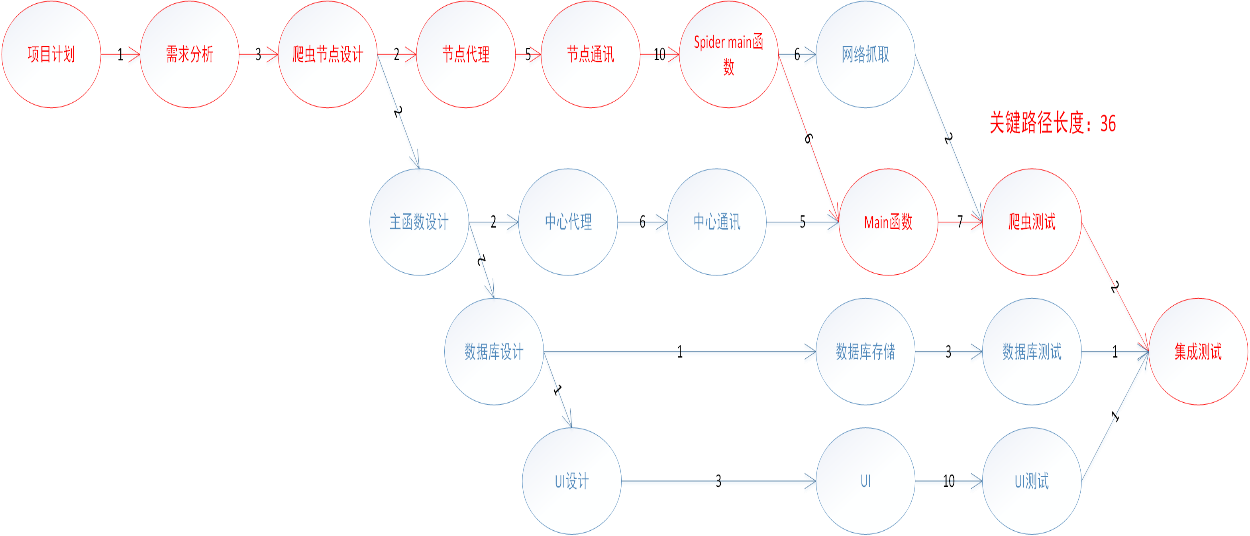


图4.1 关键路径

**分布式爬虫系统使用说明书**



一、概述：

分布式爬虫系统主要用于爬取电商网站，如京东，淘宝，天猫，网易严选，对海量的分散的电商网站数据进行采集的系统，并且展示爬取的商品信息，店铺信息和商品价格，以及实时展示爬虫系统的性能。

二、主要结构及工作原理：

1.主要结构

分布式爬虫系统由通讯模块，爬虫模块，前端模块，数据库模块组成。

2.工作原理

用户在前端web界面输入待提取的电商网页，点击开始提取，并可以在页面上看到相应电商网站商品的信息和爬虫性能数据。

三、主要性能参数：

1. 输入URL参数：

<http://www.taobao.com,>

<http://www.jd.com,>

<http://www.tianmao.com,>

http://you.163.com；

2、编写语言：html，python；

3、工作环境：windows;

四、操作指南：

1.安装Python3模块

https://jingyan.baidu.com/article/e73e26c0bad76224acb6a766.html

2.安装phantomjs模块

https://segmentfault.com/a/1190000009020535

3.安装依赖

pip install -r requirements.txt

1. 安装Chrome/Firefox浏览器
2. 开启code/main\_server/server\_main.py还有code/main\_server/spider\_main.py 。
3. 开启浏览器后打开离线web页面：login.html 输入url后点击提交即可开始采集 。
4. 页面信息隔段时间自动刷新一次。
5. 中间会有多次弹框，属正常现象。
6. 使用中略有报错，属正常现象，不影响使用。

**分布式爬虫系统总结**



1. **关键技术**

1.1反爬虫机制

1. 伪造请求头
2. 设置代理

1.2通讯机制

1. 浏览器和后台用websocket通讯，后台使用tornado接收数据
2. 服务器之间用socket通讯（含阻塞型和非阻塞型），使用了socketserver，可以每秒钟处理上千次请求。

1.3爬虫模块

1. 用phantomjs模拟浏览器采集动态网页的内容。
2. 用requests采集静态网页内容
3. 用面向对象方法封装一系列操作

1.4 UI

1. Echarts动态实时刷新图表
2. 基于websocket接收数据
3. 使用js脚本动态刷新数据

1.5 分布式技术

1. redis分布式数据库
2. 分布式去重
3. 监控爬虫子节点的内存占用和cpu使用率
4. 主--从任务分配
5. **创新点**

2.1.自己研发的后台展示系统。

2.2.自己研发爬虫模块。

2.3.自制爬虫框架。

2.4.自制通讯模块。

1. **难点以及解决方法**

3.1.前端数据动态实时刷新图表。

解决方案：

3.2.分布式数据库的汇总。

解决方案：写shell脚本或者windos批处理脚本实现远程的访问控制，将分节点的数据统一发送到主机服务器。

3.3.免费代理数量有限。

解决方案：通过shadowsocks远程连接国外的vps，通过Tor洋葱网络获取更多可用的免费代理资源。

3.4.多进程端口冲突。

解决方案：使用docker容器进行端口映射，避免冲突。但是开发平台使用windows不支持docker，切换平台。

3.5.phantomjs没有提供代理切换功能。

解决方案：通过执行js代码实现切换，但是无法切换UA。

3.6.中文编码在传输中会被转码。

解决方案：在反序列化的过程中使用decode/encode方法进行编码解码的转换。

3.7.在前端和后端交互过程中对json进行反序列化的过程中容易出现任意代码执行漏洞。

解决方案：不使用eval函数而使用json.parse()函数。

**分布式爬虫系统测试文档**



1. **模块测试**

1.1 前端模块

1.1.1 前端与后端通讯测试

前端初始页面和后端代码通讯，将输入的URL通过websocket传输给后端python代码，前端初始页面如图1.1.1-1所示。



图 1.1.1-1 初始页面

依次输入

https://www.taobao.com

<http://www.jd.com.>

<https://www.tmall.com>

<http://you.163.com>

在后端接收如图1.1.1-2，1.1.1-3，1.1.1-4，1.1.1-5所示。

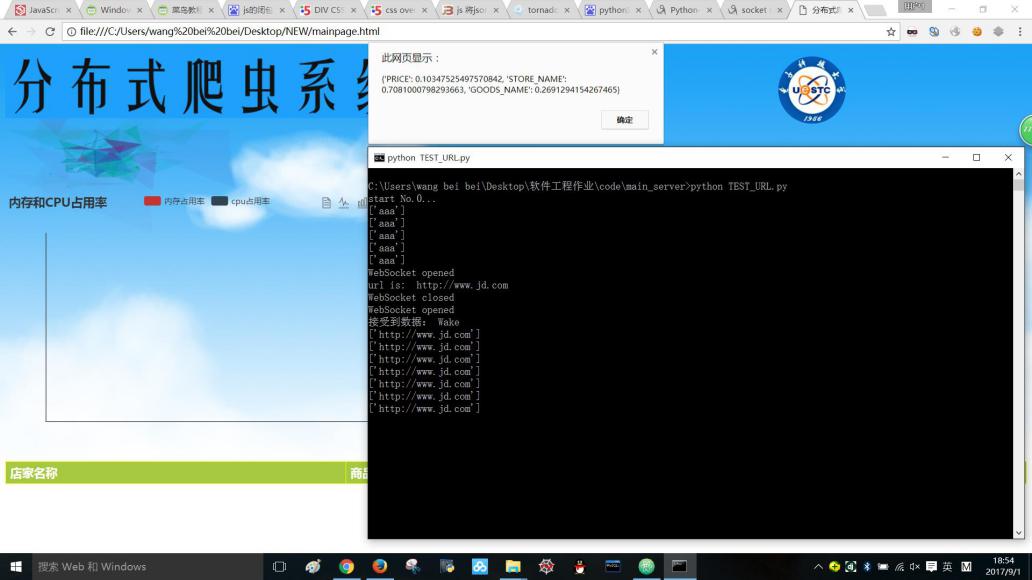


图1.1.1-1 后台收到数据展示

1.1.2 前端动态图表测试

（1）动态柱状图测试

对动态的柱状图的测试，模拟数值输入，模拟数值如下：

内存占用率：data:[2.0, 4.9, 7.0, 23.2, 25.6, 76.7, 135.6, 162.2, 32.6, 20.0, 6.4, 3.3],

Cpu利用率：data:[2.6, 5.9, 9.0, 26.4, 28.7, 70.7, 175.6, 182.2, 48.7, 18.8, 6.0, 2.3],

测试数据输入后得到的动态柱状图如图1.1.2-1所示。

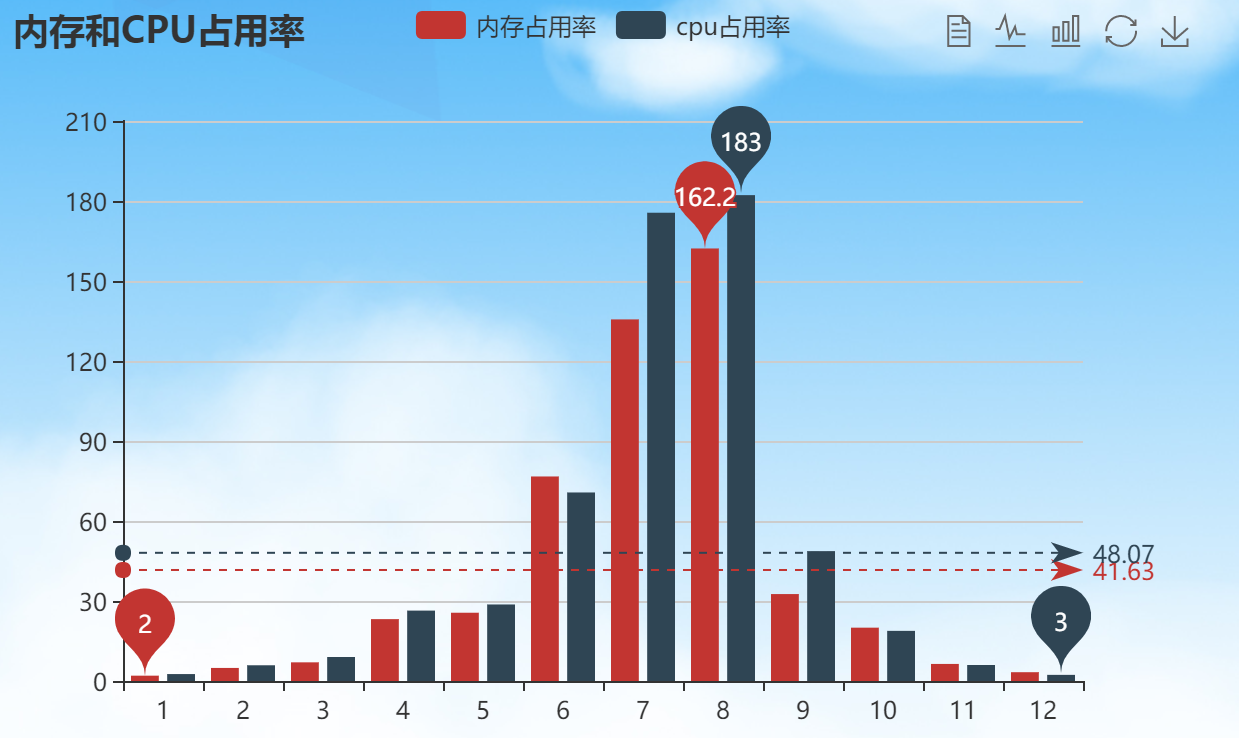


图1.1.2-1 动态柱状图测试

（2）动态折线图测试

对动态的折线图的测试，用随机模拟数据输入进行测试，随机数模拟代码如1.1.2-2所示。

function randomData() {

now = new Date(+now + oneDay);

value = value + Math.random() \* 21 - 10;

return {

name: now.toString(),

value: [

[now.getFullYear(), now.getMonth() + 1, now.getDate()].join('/'),

Math.round(value)

]

}

}

1.1.2-2 随机模拟输入折线图代码

var data = [];

var now = +new Date(1997, 9, 3);

var oneDay = 24 \* 3600 \* 1000;

var value = Math.random() \* 1000;

for (var i = 0; i < 1000; i++) {

data.push(randomData());

}

输入随机数以后，折线图显示如图1.1.2-3所示。

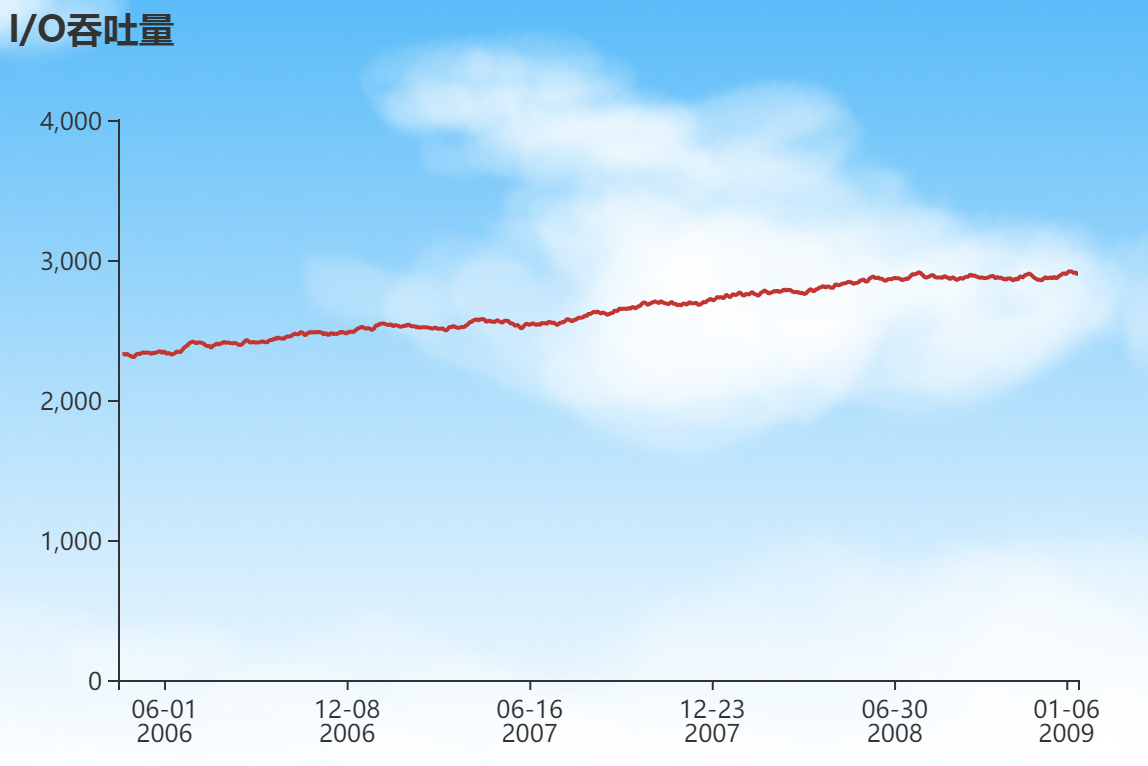


图1.1.2-3 动态折线图测试效果

（3）动态页表页测试

对动态页面测试，输入模拟数据：stack=["¥1699.00","NIKE轻便跑鞋","NIKE全网专卖","1099.00","NIKE篮球鞋","NIKE全网专卖","NOTHOMME","NOTHOMME日系潮牌肌理印花连帽套头卫衣男款宽松版街头潮牌上衣白","¥168.00",];输入动态数据后，页面显示如图1.1.2-4所示。

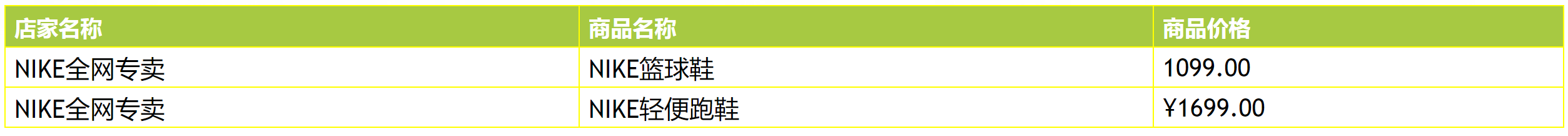


图1.1.2-4动态图表测试效果图

1.2 通讯模块

1.2.1 主机和节点通讯

主机通过TCP协议和节点进行通信，主要交换ip、端口和URL数据，通讯模块由TCP\_sender 和TCP\_reciever函数，以及以此为基础构建的位于主机的sending\_ip\_and\_url.py 以及位于节点的recieving\_ip\_and\_port.py构成。接下来我们对通讯功能的主体sending\_ip\_and\_url（）函数

和recieving\_ip\_and\_port（）进行测试，测试代码显示如图1.2.1-1所示

|  |
| --- |
| #主机代码：  #!/usr/bin/python  # -\*- coding: utf-8 -\*-  from server\_ip\_and\_port.sending\_ip\_and\_url import sending\_ip\_and\_url  sending\_ip\_and\_url("127.0.0.1",8880,["127.0.0.1","127.0.0.2"],[80,90],"aaa")  input()  #节点代码：  #!/usr/bin/python  # -\*- coding: utf-8 -\*-  from spider\_ip\_and\_port.receiving\_ip\_and\_port import receiving\_ip\_and\_port  print(receiving\_ip\_and\_port('127.0.0.1',8880))  #返回接受的信息  input() |

图1.2.1-1 通讯模块测试代码

测试结果如图1.2.1-2所示，节点成功接收主机发送的ip和端口数据并打印，，通讯模块功能测试通过，各函数也未抛出任何异常。

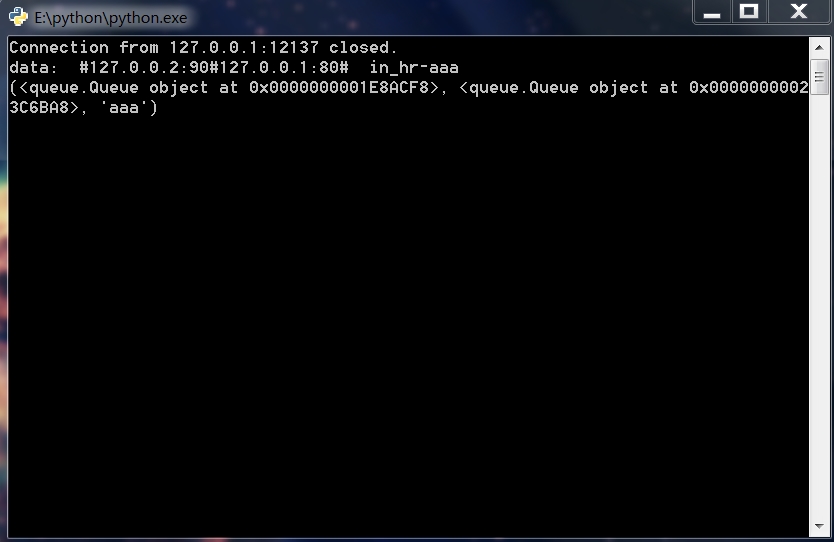


图1.2.1-2 通讯模块测试结果

1.2.2 浏览器和主机通讯

主机爬取数据以后，通过websocket将数据传输给前端，连接测试代码

如1.2.2-1所示。

|  |
| --- |
| <!DOCTYPE html>  <meta charset="utf-8" />  <title>WebSocket Test</title>  <script language="javascript"type="text/javascript">  var wsUri ="ws://113.54.248.185:10008/";  //var wsUri ="ws://192.168.1.102:10000/url";  var output;  function init() {  output = document.getElementById("output");  testWebSocket();  }  function testWebSocket() {  websocket = new WebSocket(wsUri);  websocket.onopen = function(evt) {  onOpen(evt)  };  websocket.onclose = function(evt) {  onClose(evt)  };  websocket.onmessage = function(evt) {  onMessage(evt)  };  websocket.onerror = function(evt) {  onError(evt)  };  }  function onOpen(evt) {  writeToScreen("CONNECTED");  doSend("WebSocket rocks");  }  function onClose(evt) {  writeToScreen("DISCONNECTED");  }  function onMessage(evt) {  writeToScreen('<span style="color: blue;">RESPONSE: '+ evt.data+'</span>');  var a=evt.data;  var b=eval('(' + a + ')');  websocket.close();  }  function onError(evt) {  writeToScreen('<span style="color: red;">ERROR:</span> '+ evt.data);  }  function doSend(message) {  writeToScreen("SENT: " + message);  websocket.send(message);  }  function writeToScreen(message) {  var pre = document.createElement("p");  pre.style.wordWrap = "break-word";  pre.innerHTML = message;  output.appendChild(pre);  }  window.addEventListener("load", init, false);  </script>  <h2>WebSocket Test</h2>  <div id="output"></div>  </html> |

1.2.2-1 websocket连接测试代码

测试的效果图如图1.2.2-2所示。

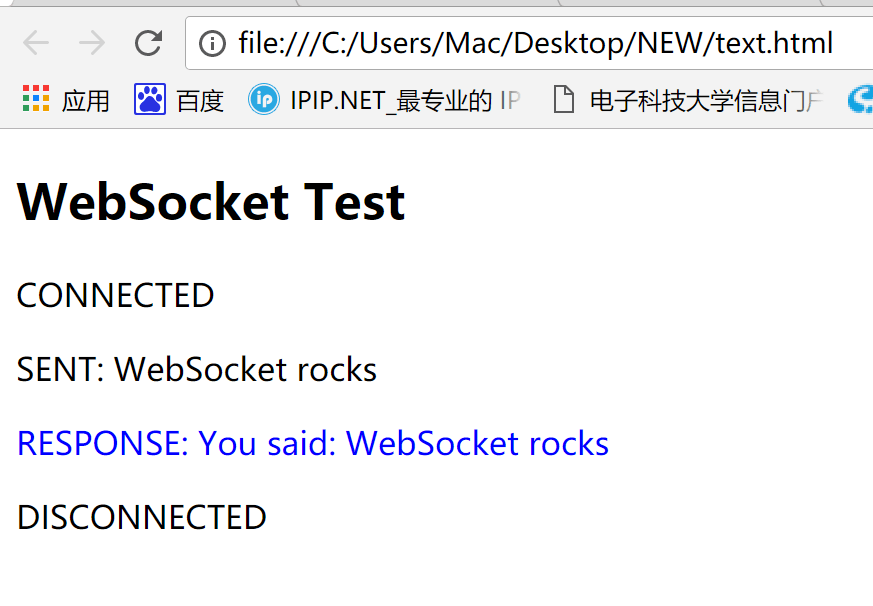


图1.2.2-2 websocket连接测试成功图

建立连接后，用python模拟数据向前端传输，并且分析传输的数据，测试结果如图1.2.2-3所示。

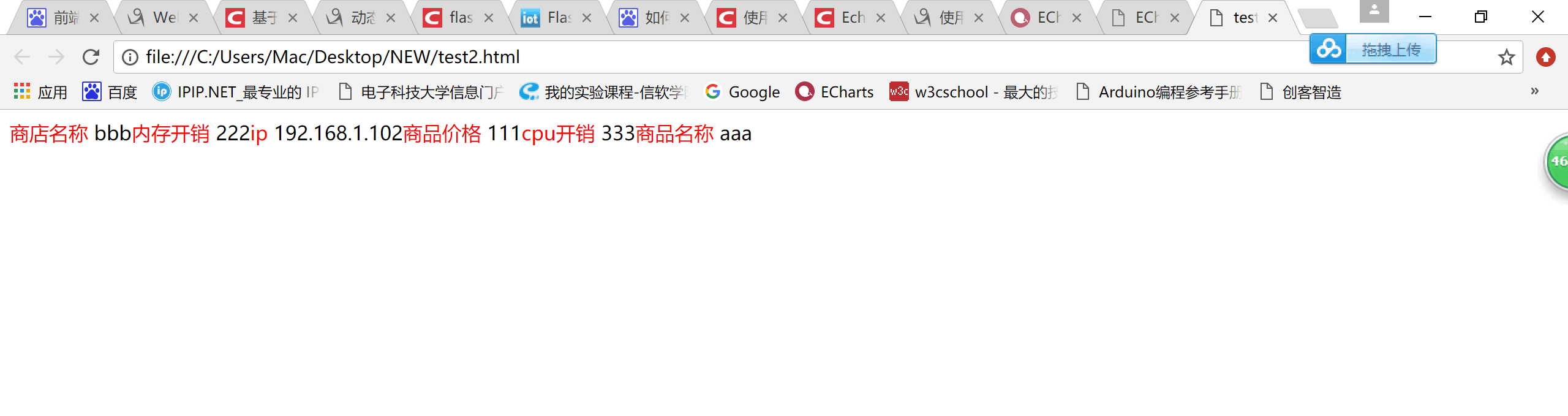


图1.2.2-3 数据传输分析成功图

1.3 爬虫模块

1.3.1 URL去重

分析URL处理函数，如图1.3.1-1所示，选择一串部分重复的URL作为输入，输出结果如1.3.1-2，成功去除重复的URL，测试通过

|  |
| --- |
| def processing\_the\_url(data):  global set\_,url\_queue  #global unrepeated\_url\_io  if data in set\_:  q=data+" repeated"  #lock.wait(t\_num)  #print("接受到重复的url: ",q)  url\_queue.put(q)  """  try:  conn.sendall(q.encode())  except Exception as e:  print(e)  """  #lock.signal(t\_num)  else:  q=data+" unrepeated"  #print("接受到未重复的url: ",q)  #print('\n吞吐量计数: ',unrepeated\_url\_io,'\n')  #lock.wait(t\_num)  set\_.add(data)  url\_queue.put(q) |

1.3.1-1 URL处理函数

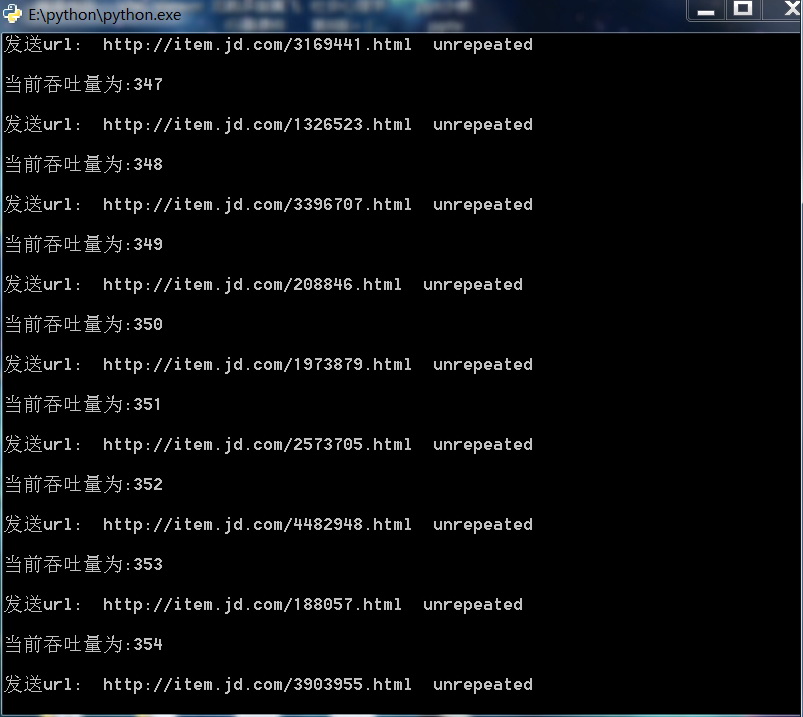


图1.3.1-2 URL去重结果

1.3.2 模拟浏览器请求模块

利用selenium发起模拟浏览器请求，嵌入在其他函数内部进行测试

|  |
| --- |
| def get\_url\_and\_post\_them  (remoted\_ip='127.0.0.1',url\_sending\_port=9100,url='',spider\_ip=[]):  #区分内外链tmail.com与jd.com  ....  driver=webdriver.PhantomJS()  driver.get(url)  .... |

测试结果如图1.3.2-1所示，成功发起模拟浏览器请求。

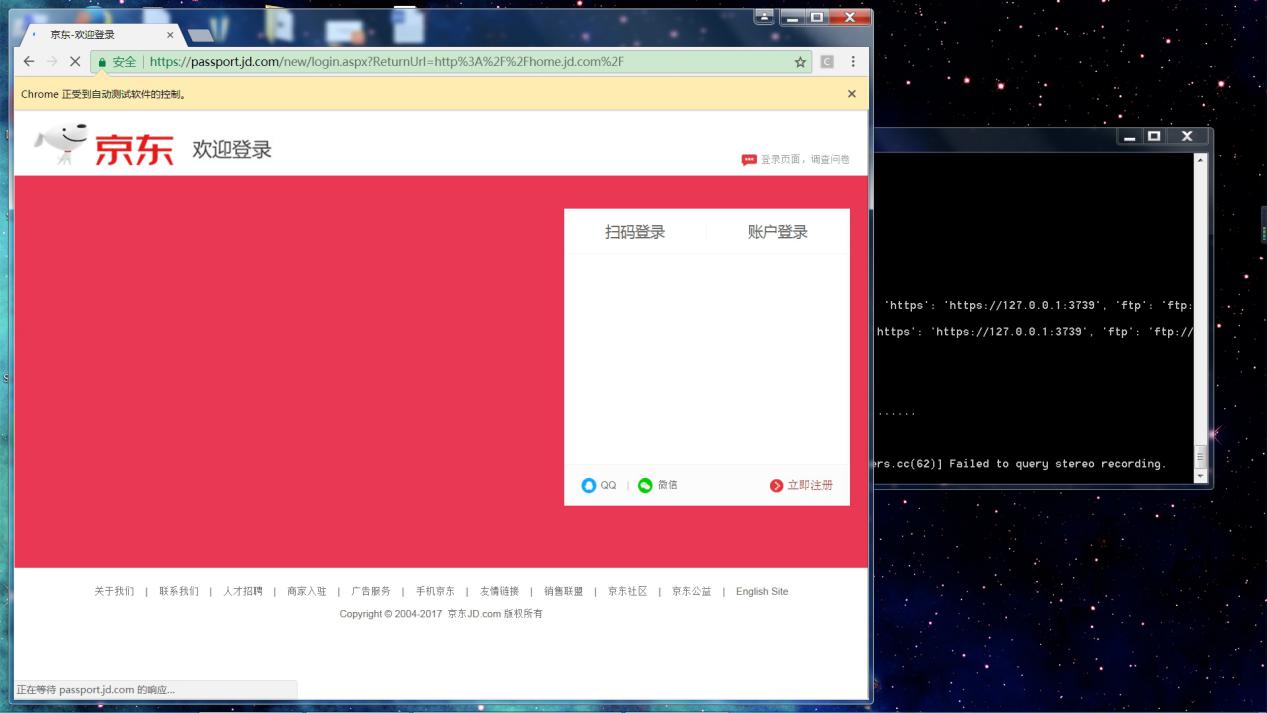


图1.3.2-1 模拟浏览器请求测试

1.3.3 反爬功能

反爬功能由phantomjs模拟浏览器+免费匿名代理、

fakeuseragent库使用伪造的useragent 、类中自带的定期切换代理 三个机制实现，对免费匿名代理进行测试，结果如图1.3.3-1所示

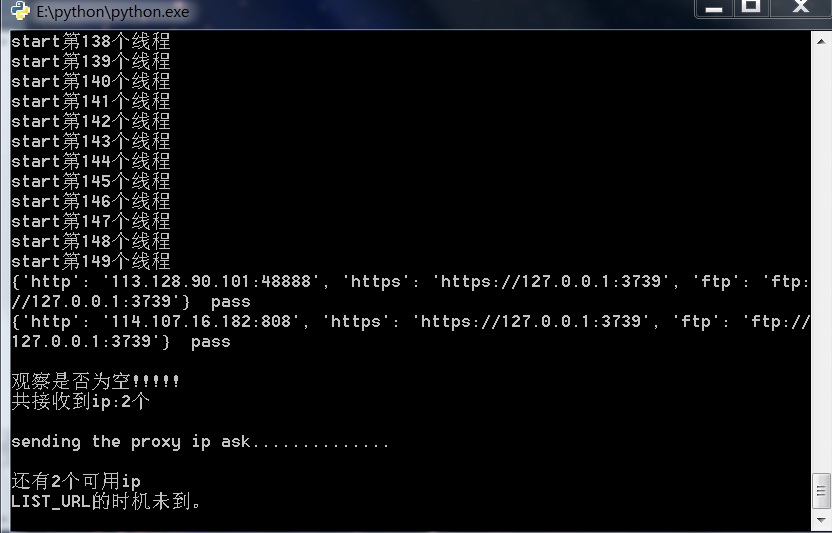


图1.3.3-1 测试收到的代理ip个数

1.3.4 网页自动结构化

对网页进行自动结构化，测试结果如图1.3.4-1所示，成功结构化并抽取商品数据。



图1.3.4-1 结构化电商数据

最后对爬虫模块进行综合测试，图1.3.4-2所示，各函数协作正常，按预期效果进行，未抛出异常，爬虫模块测试成功

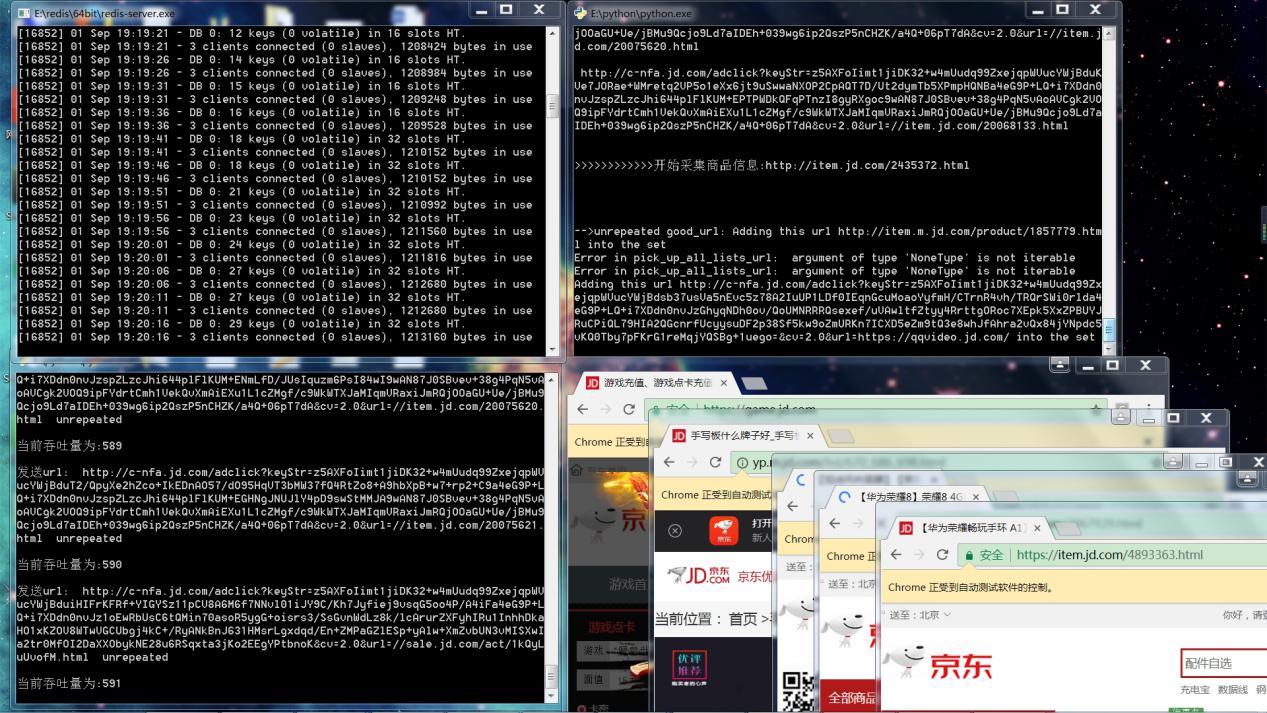


图1.3.4-2 爬虫全流程测试

1.4 数据库模块

1.4.1 redis操作函数测试

Redis\_module中定义了三种操作，连接，存储，读取操作，现在对其进行测试，测试代码如1.4.1-1所示。

|  |
| --- |
| #!/usr/bin/env python  # -\*- coding:utf-8 -\*-  from redis\_module.redis\_store import database  d=database()  d.store('goods\_name\_testcase',"store\_name\_testcase",'goods\_price\_testcase')  d.store('gold medal',"UESTCer's markert",'80000')  d.store('计算机',"UESTCer's markert",'80000')  d.read()  input() |

1.4.1-1 数据库操作测试代码

测试结果如图1.4.1-2所示，从上至下分别是：测试代码返回的在list中做主键的商品名、redis\_server、redis\_cli中进一步查看店铺名和价格

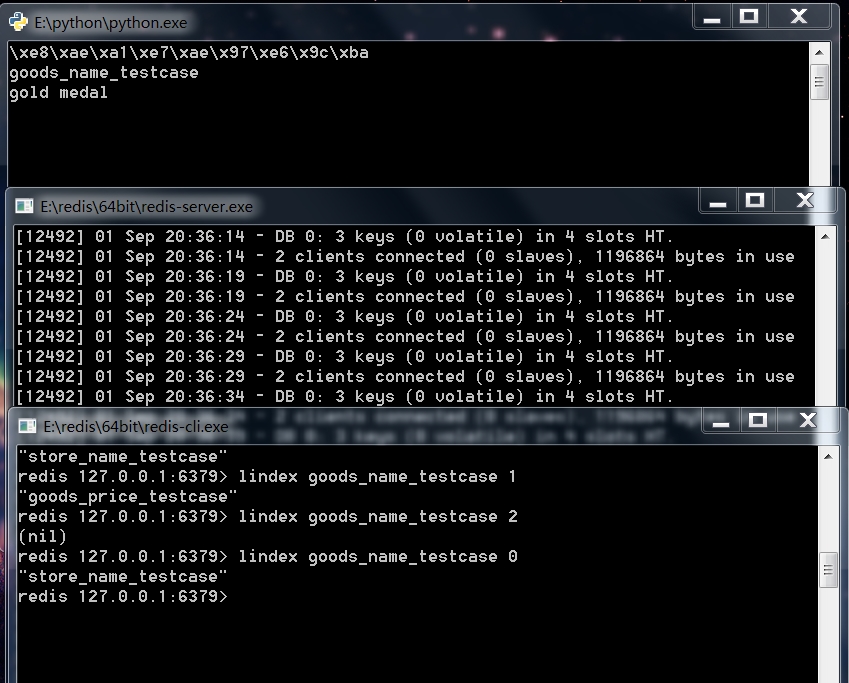


图1.4.1-3 数据库测试结果

1.4.2 数据库存储功能测试

为了验证数据库存储模块作为分布式爬虫系统的一个功能时的正确性，我们还需要进行模块功能测试，直接运行后端系统，测试结果如1.4.2-1所示，成功抓取到‘澳柯玛’冰箱的数据，并以十六进制的方式存储在redis数据库中，

如1.4.2-2所示。数据库存储功能测试通过。



图1.4.2-1 成功爬取并存储数据

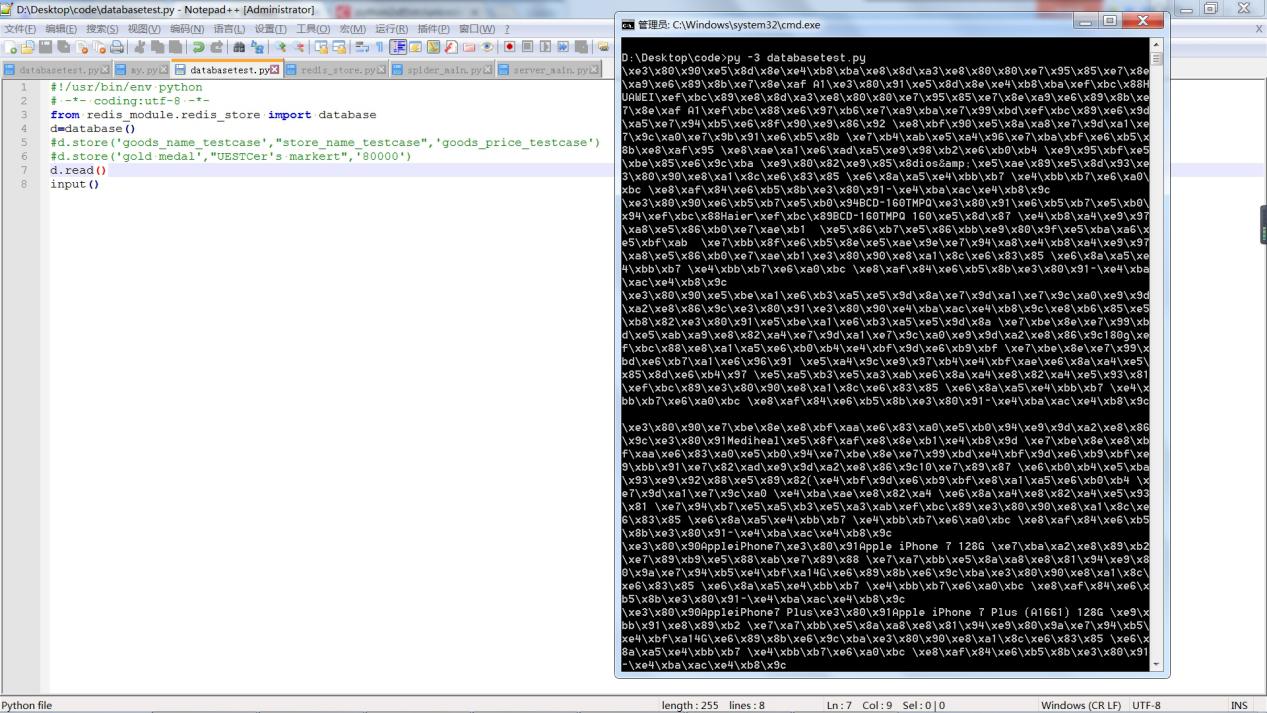


图1.4.2-2 查看数据库中的数据

**分布式爬虫系统作品简介**

**与参赛感言**



1. **作品简介**

**1.1 作品名称**

分布式爬虫系统

**1.2 作品作者**

电子科技大学iMake小组，成员：郑毅，王骞，祝含颀；

指导教师：吴晓华教授。

**1.3 作品功能**

对电商网站如淘宝，天猫，京东，网易严选中海量商品数据的爬取和数据分析。实现反爬虫机制，网页自动结构化，爬取速度快而稳定。

**1.4 作品技术**

1.4.1反爬虫机制

1. 伪造请求头
2. 设置代理

1.4.2通讯机制

1. 浏览器和后台用websocket通讯，后台使用tornado接收数据
2. 服务器之间用socket通讯（含阻塞型和非阻塞型），使用了socketserver，可以每秒钟处理上千次请求。

1.4.3爬虫模块

1. 用phantomjs模拟浏览器采集动态网页的内容。
2. 用requests采集静态网页内容
3. 用面向对象方法封装一系列操作

1.4.4 UI

1. Echarts动态实时刷新图表
2. 基于websocket接收数据
3. 使用js脚本动态刷新数据

1.4.5 分布式技术

1. redis分布式数据库
2. 分布式去重
3. 监控爬虫子节点的内存占用和cpu使用率
4. 主--从任务分配

**1.5 作品创新和实现的功能**

1.5.1 创新点

（1）自己研发的后台展示系统。

（2）自己研发爬虫模块。

（3）自制爬虫框架。

（4）自制通讯模块。

（5）实现了分布式的节点调度和分布式的数据存储，分节点监听。

1.5.2 实现的功能

1. 爬取数据页面展示
2. 反爬虫机制
3. 爬取速度快
4. 网页自动结构化，URL去重调度。
5. **参赛感言**

队员一:

一开始我们对软件比赛完全没有概念，我们小组也是在一门专业课上组建的，老师鼓励我们自主寻找课题，后来组内的一个同学提议“中国软件杯”中的课题，这才开始逐渐了解比赛的各个方面。所以我们主要还是抱着学习的心态来参加比赛的，希望更好地加强自己的实践能力和团队合作能力。

队员二：

准备比赛过程中我最大的感受，就是只有亲自动手，才能体会到理论是如何指导实践的。平时所学看似简单的理论，实践起来并不是那么轻松，一个小小的代码都会导致整个系统的崩溃，更何况还有很多知识都是平时未曾涉及的。

队员三：

我觉得在比赛过程中从其他参赛队伍中学到了很多。无论获奖与否，各个队伍对同一问题的不同解决方式让我大开眼界。当然最关键的还是指导老师从始至终的鼓励，最大的收获还是培养了团队合作精神。

此次比赛，我们队最看重的是动手实践，对于这样开源性的赛题，有很多开源的框架，但是我们并不愿意去用，因为有很多技术并不是止步在现有的技术，我们需要自己去探索，自己去发现现有技术的不足，而不是做一些没有用的外包技术，简单的重叠代码。