**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。

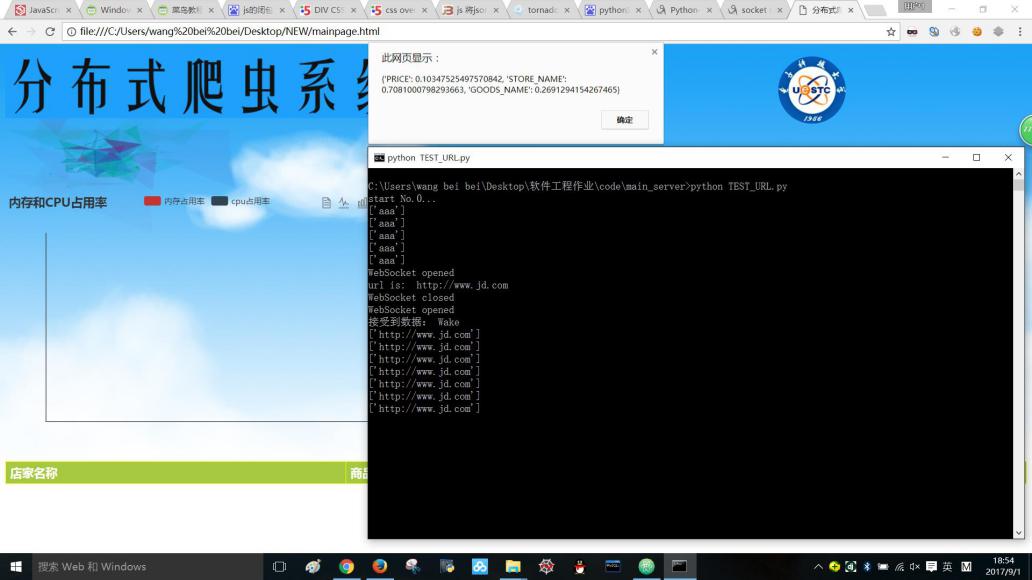


图1.1.1-1 后台收到数据展示

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。

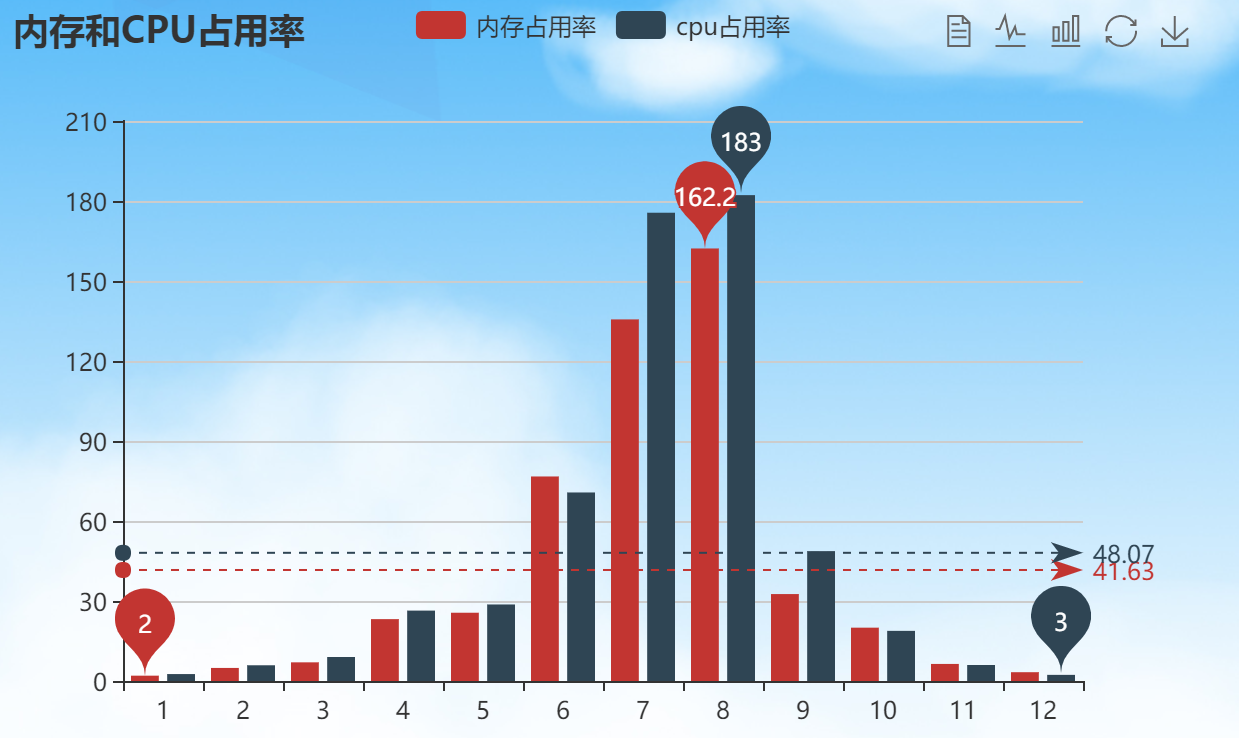


图1.1.2-1 动态柱状图测试

（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。

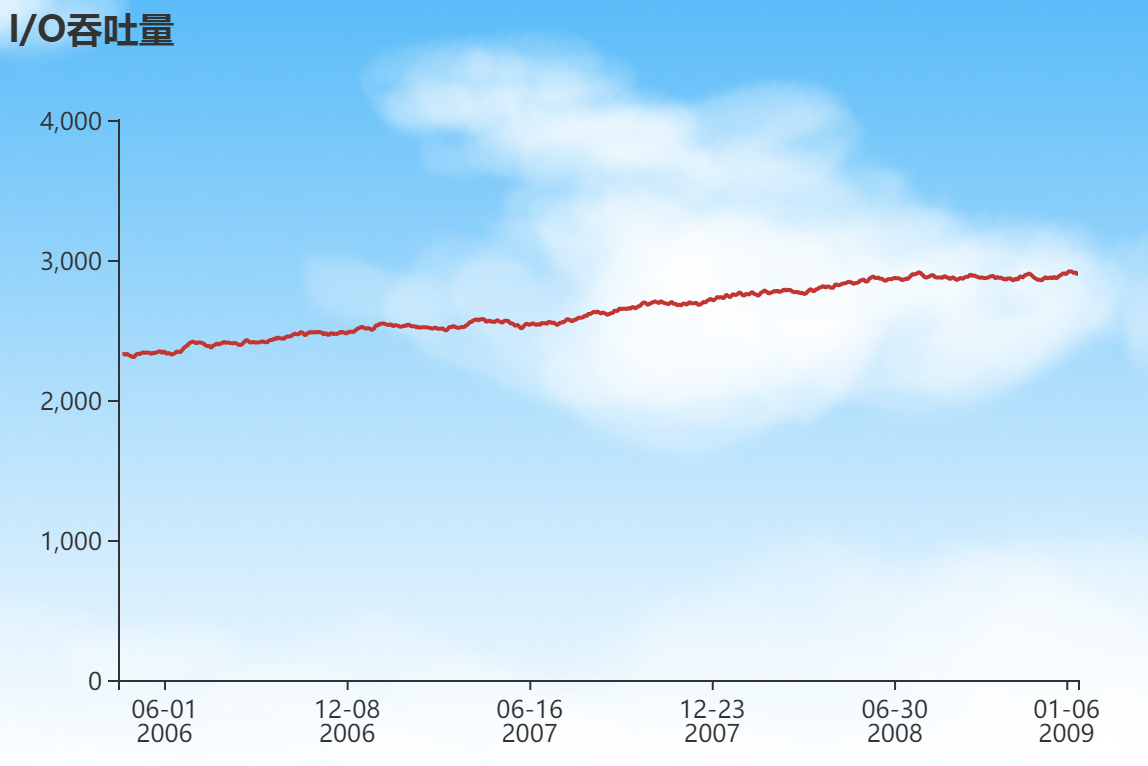


图1.1.2-3 动态折线图测试效果

（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。

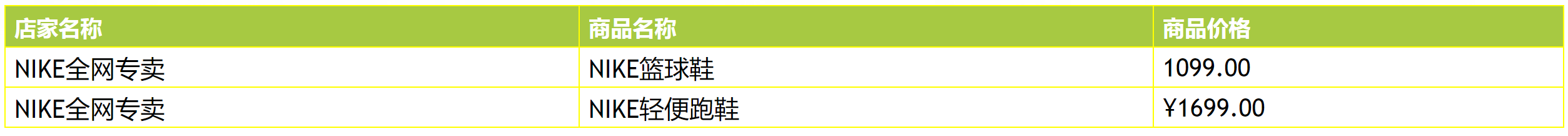


图1.1.2-4动态图表测试效果图

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。

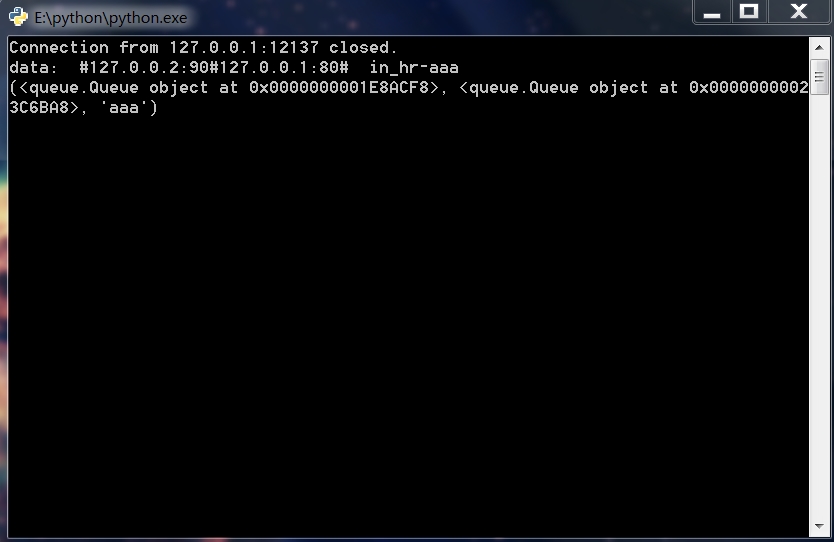


图1.2.1-2 通讯模块测试结果

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。

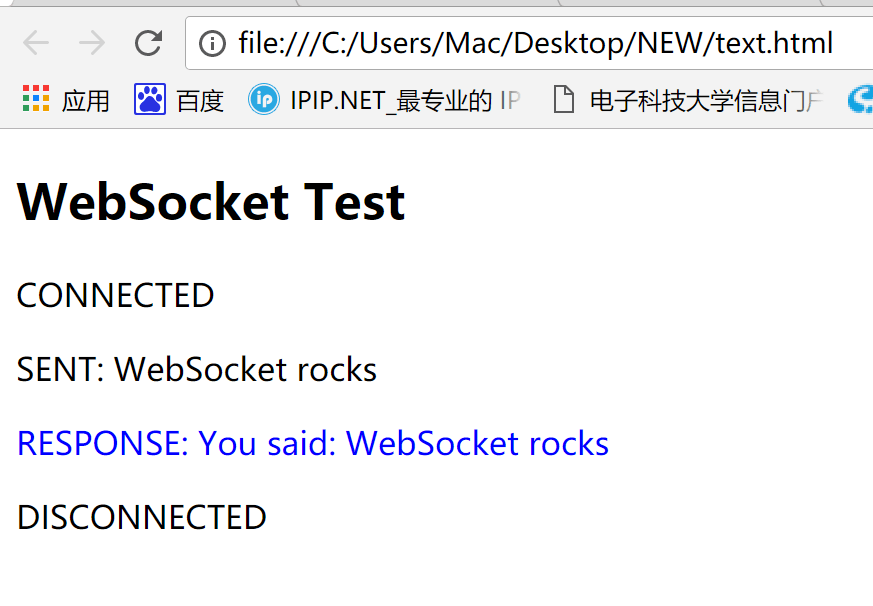


图1.2.2-2 websocket连接测试成功图

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。

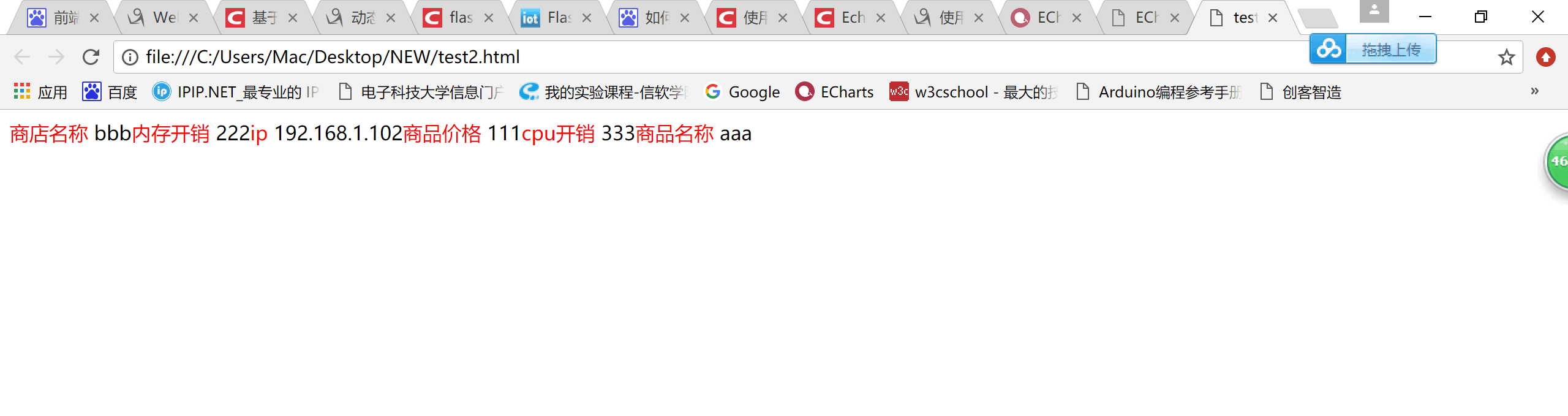


图1.2.2-3 数据传输分析成功图

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

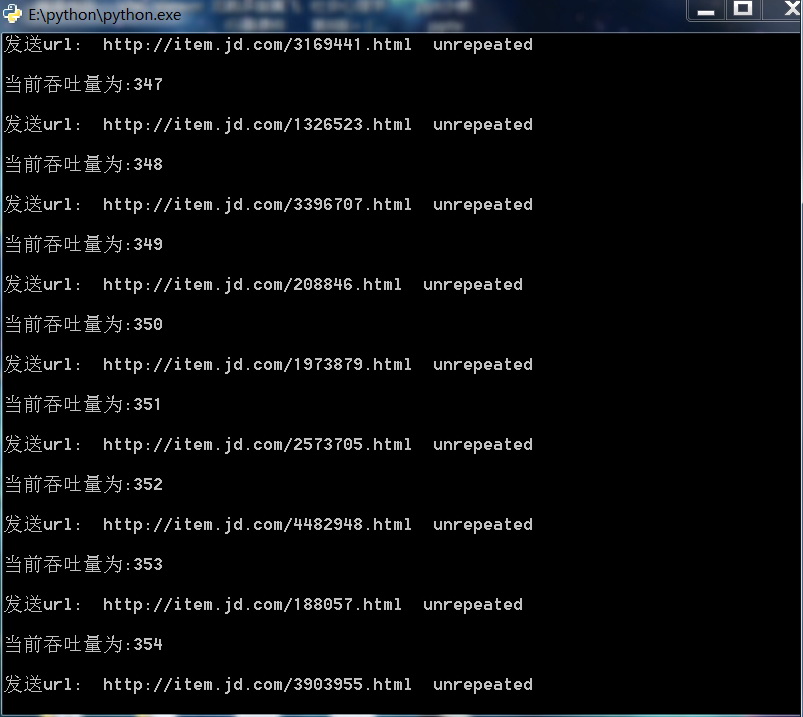


图1.3.1-2 URL去重结果

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。

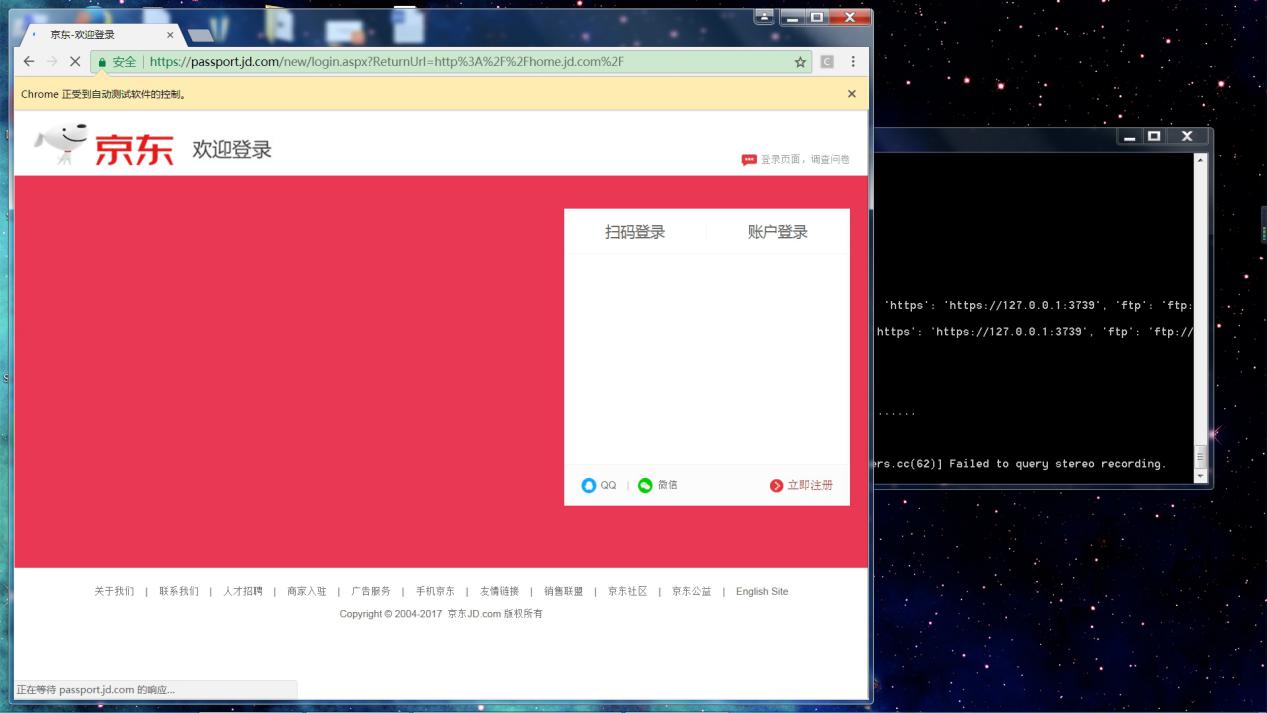


图1.3.2-1 模拟浏览器请求测试

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

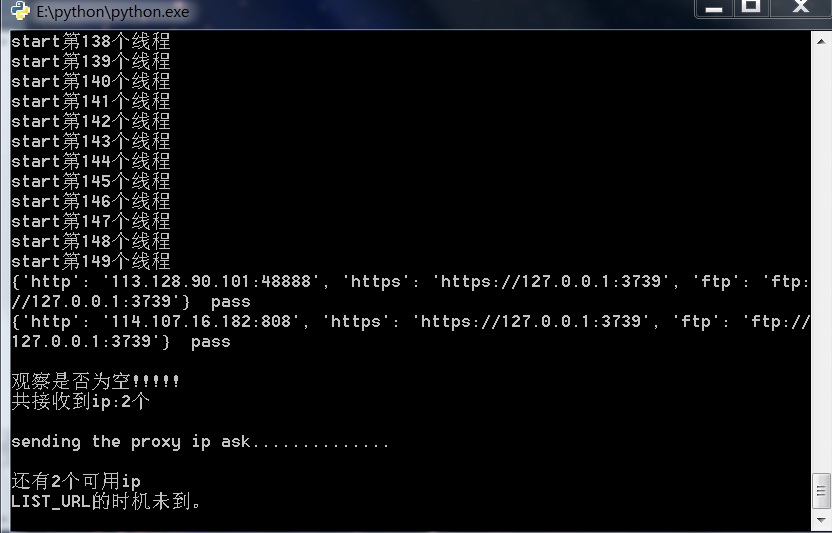


图1.3.3-1 测试收到的代理ip个数

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

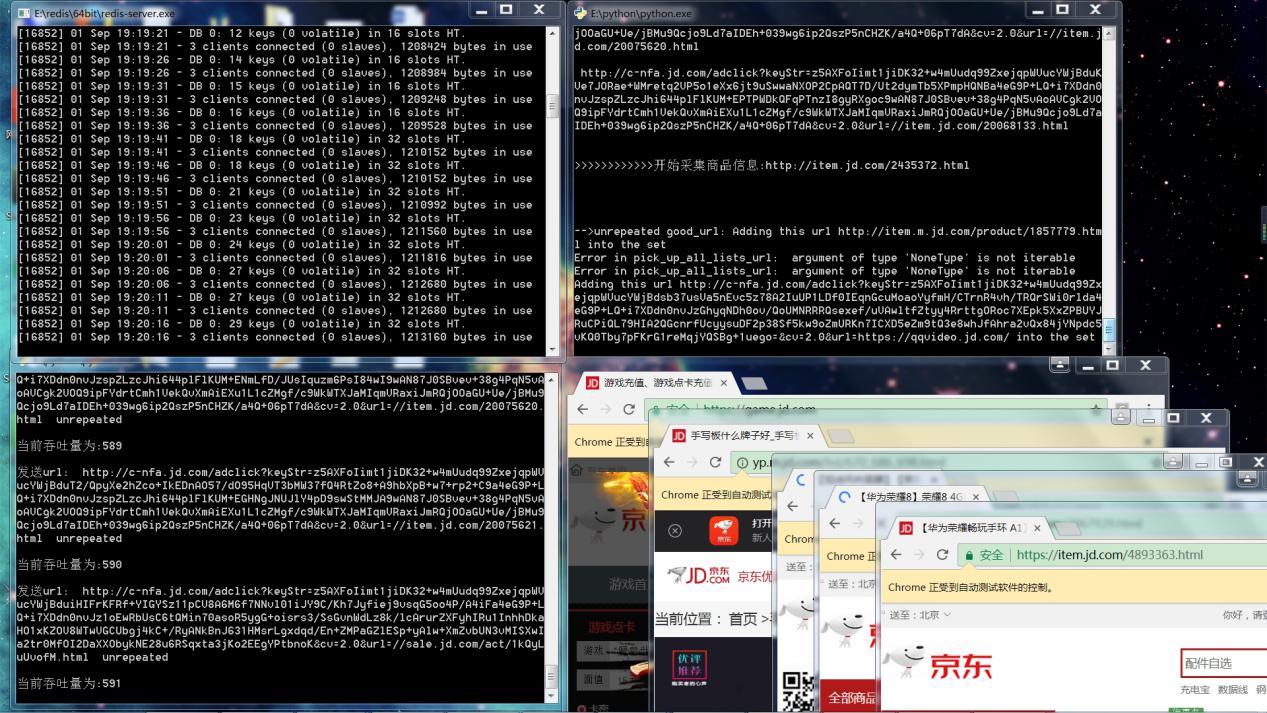


图1.3.4-2 爬虫全流程测试

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

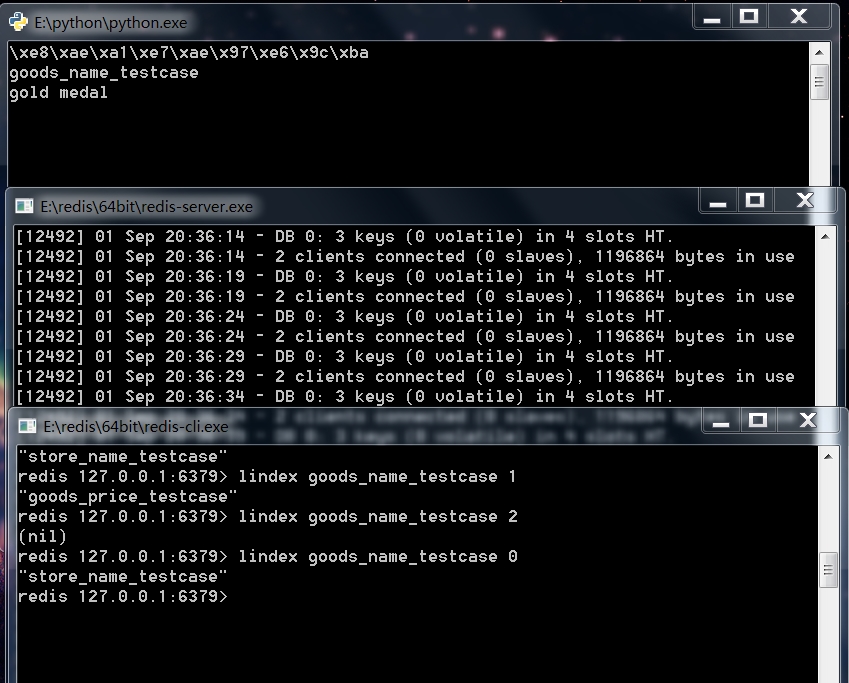


图1.4.1-3 数据库测试结果

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

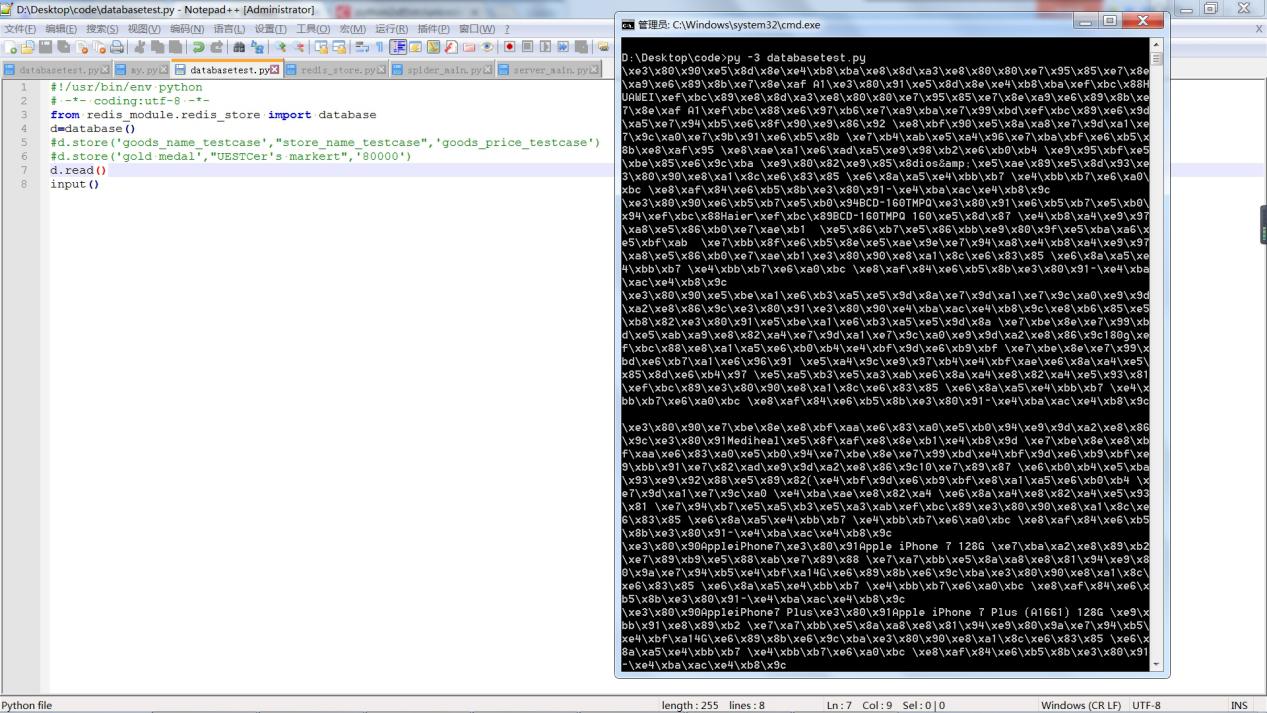


图1.4.2-2 查看数据库中的数据