浙江移动云爬平台

用户手册

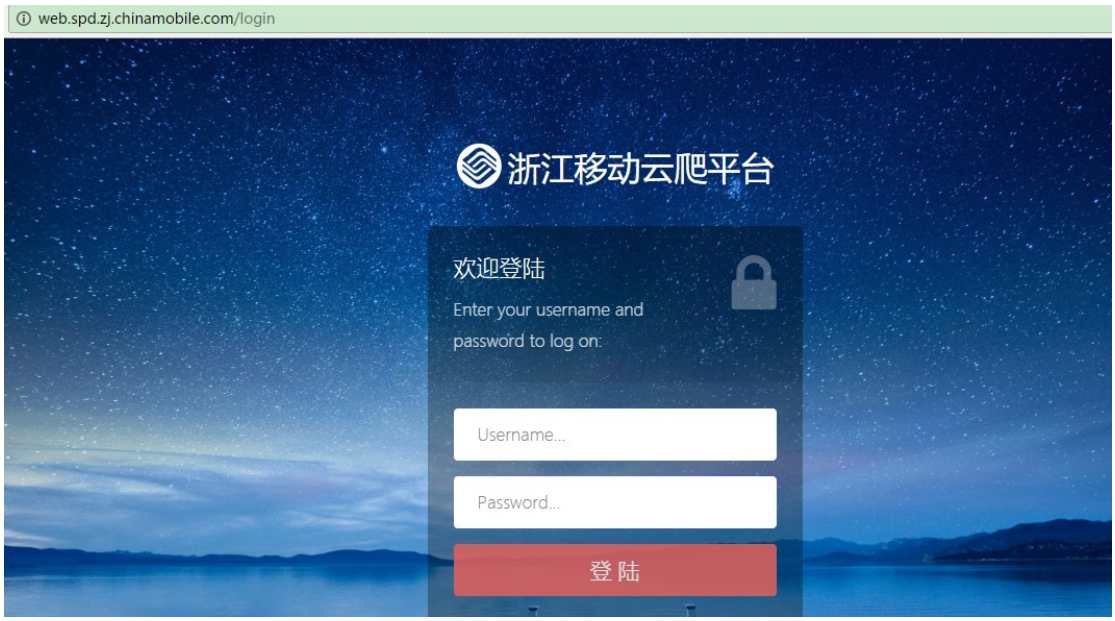
（v1.0 版）

大数据中心

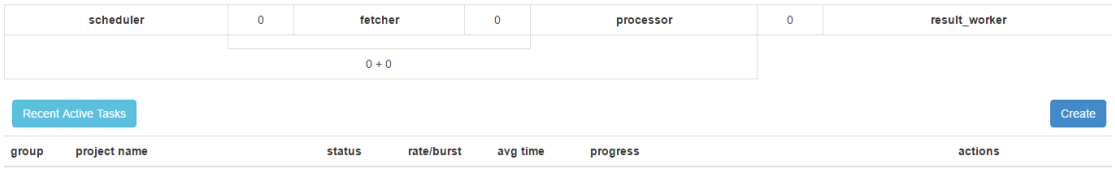
目录

一、平台入口

http://web.spd.zj.chinamobile.com/login



输入用户名、密码 点击登录



二、快速入门

2.1 关于架构

2.1.1 前言

pyspider 是一个爬虫架构的开源化实现主要的功能需求是：

1).抓取、更新调度多站点的特定的页面

2).需要对页面进行结构化信息提取

3).灵活可扩展，稳定可监控

而这也是绝大多数 python 爬虫的需求 —— 定向抓取，结构化化解析。但是面对结构迥异的各种网站，单一的抓取模式并不一定能满足，灵活的抓取控制是必须的。为了达到这个目的，单纯的配置文件往往不够灵活，于是，通过脚本去控制抓取是我最后的选择。而去重调度，队列，抓取，异常处理，监控等功能作为框架，提供给抓取脚本，并保证灵活性。

最后加上 web 的编辑调试环境，以及 web 任务监控，即成为了这套框架。

pyspider 的设计基础是：以 python 脚本驱动的抓取环模型爬虫

1)．通过 python 脚本进行结构化信息的提取，follow 链接调度抓取控制，实现最大的灵活性；

2)．通过 web 化的脚本编写、调试环境。web 展现调度状态；抓取环模型成熟稳定，模块间相互独立;

3)．通过消息队列连接，从单进程到多机分布式灵活拓展

pyspider

2.1.2 功能

webui

1). web 的可视化任务监控

2). web 脚本编写，单步调试

3). 异常捕获、log 捕获，print 捕获等

scheduler

1). 任务优先级

2). 周期定时任务

3). 流量控制

4). 基于时间周期 或 前链标签（例如更新时间）的重抓取调度

fetcher

1). dataurl 支持，用于假抓取模拟传递

2). method, header, cookie, proxy, etag, last_modified, timeout 等等抓取调度控制

3). 可以通过适配类似 phantomjs 的 webkit 引擎支持渲染

processor

1). 内置的 pyquery，以 jQuery 解析页面

2). 在脚本中完全控制调度抓取的各项参数

3). 可以向后链传递信息

4). 异常捕获

result_work

处理结果的爬虫组件，结果入库

2.1.3 架构

pyspider 的架构主要分为 scheduler（调度器）, fetcher（抓取器）, processor（脚本执行）：

1). 各个组件间使用消息队列连接，除了 scheduler 是单点的，fetcher 和 processor 都是可以多实例分布式部署的。 scheduler 负责整体的调度控制；

2). 任务由 scheduler 发起调度，fetcher 抓取网页内容， processor 执行预先编写的 python 脚本，输出结果或产生新的提链任务（发往 scheduler），形成闭环。

3). 每个脚本可以灵活使用各种 python 库对页面进行解析，使用框架 API 控制下一步抓取动作，通过设置回调控制解析动作。

2.2 项目（Projects）

大多数情况下，一个项目就是针对一个网站写的一个爬虫脚本。

1). 项目是相对独立的但是你可以导入其它项目或其它项目的模块.

2). 项目有五个状态:TODO,STOP,CHECKING,DEBUG,RUNNING

TODO- 当一个脚本刚刚被创建时的状态

STOP- 设置项目状态为 STOP 让项目停止运行

CHECKING- 当一个运行中的项目被编辑时项目状态会被自动设置成此状态并停止运行.

DEBUG/RUNNING- 这两状态都会运行爬虫，但是他们之间是有区别的。一般来说调试阶段用 DEBUG 状态，线上用 RUNNING 状态。

3). 爬虫的抓取速度根据网上流行的 token-bucket 来控制。

rate- 每秒执行多少个请求。

burst- 设置并发数,如：rate/burst = 0.1/3,这个的意思是爬虫 10 秒爬一个页面。但是开始时前三个任务会同时时行，不会等 10 秒，第四个任务爬取前会等 10 秒。

4). 项目删除：把 group 设置成 delete 并把项目状态设置成 STOP，２４小时后系统会自动删除此项目

2.3 任务

任务是调度器调度的最基本单元

2.3.1 任务属性

一个任务有唯一的任务 ID 叫 taskid(默认是：md5(url),不过可以重写 get_taskid(self,task)方法指定自己项目生成 taskid 方法。)

不同项目之间任务不冲突。

任务的四个状态:active（运行）、failed（失败）、success（成功）、bad - （暂时没用着）只有当任务状态为运行（active）时才会被调度。

根据优先级执行任务.

2.3.2 Schedule（调度器）

当遇到一个新任务（之前没有遇到过的链接）时：如果设置了 exetime 并且还没到时间，任务会被添加到等待队列。否则会接受添加到执行队列；

当任务已经在队列中:如果没有设置 force_update 就会忽略

当遇到一个之前爬过的任务：如果设置了 age,并且 last_crawl_age+age<now 任务会被接受并添加到执行队列，否则被忽略；如果设置了 itag 且值不等于之前的值会添加到执行队列重新爬取，否则被忽略。

2.4 环境

2.4.1 变量

self.project_name　　项目名
self.projectinformation　　当前项目信息
self.response　　响应信息
self.task　　任务信息

2.4.2 关于脚本

脚本的名称不重要，但是必需有一个类继承 BaseHandler。

每个方法默认都会传三个参数用来获取相关信息分别是：自身 self，请求信息 response，项目信息 task。如：def callback(self, response, task)

默认情况下响应代码不等于 200 的都会被忽略，但是你可以设置 @catch_status_code_error 参数来改变默认参数

2.4.3 关于环境

日志和异常都会被捕获打印输出
可以导入其它项目或项目的模块使用

2.4.4 关于视图

ＷＥＢ视图：Web view
在 web 控制台以浏览器模式显示页面内容
ＨＴＭＬ视图：HTML view
在回调函数时显示ＨＴＭＬ代码
跟踪视图：Follows view
查看可以从当前回调中获得的回调
消息视图：Messages view
显示来自 self.send_message API 接口的消息
ＣＳＳ选择器助力：Enable CSS Selector Helper
ＣＳＳ选择器助手可以在 web 视图中通过单击获取节点的 css 规则，并自动插件到脚本。

2.5 结果

在 WebUI 控制台上 Results 查看和下载爬取结果，有 json、url_json、txt 三种格式。

2.6 API 参考

2.6.1 self.crawl

self.crawl(url, **kwargs)

self.crawl 是 pyspider 系统中非常重要的接口，它的功能是告诉 pyspider 哪些 URL 需要抓取。

必选参数:

url：需要被抓取的 url 或 url 列表

callback：这个参数用来指定爬取内容后需要哪个方法来处理内容，一般解析为 response。_default: __call__ _ 如下面调用方法：

```
def on_start(self):
    self.crawl('http://scrapy.org/', callback=self.index_page)
```

可选参数：

age：用来指定任务的有效期，在有效期内不会重复抓取。默认值是-1（永远不过期，意思是只抓一次）

```
@config(age=10 * 24 * 60 * 60)
    def index_page(self, response):
        ...
```

解析：每一个回调 index_page 的任务有效期是 10 天，在 10 天之内再遇到这个任务都会被忽略掉（除非有强制抓取参数才不会忽略）.

priority：用来指定任务的优先级，数值越大越先被执行，默认值为 0。

```
def index_page(self):
    self.crawl('http://www.example.org/page2.html', callback=self.index_page)
    self.crawl('http://www.example.org/233.html', callback=self.detail_page,priority=1)
```

解析：这两个任务如果被同时放入到任务队列里，页面 233.html 先被执行。

exetime：执行任务的时间，默认值:0

```
import time def on_start(self):
    self.crawl('http://www.example.org/', callback=self.callback,exetime=time.time()+30*60)
```

解析：页面将在 30 分钟后被抓取

retries：任务执行失败后重试次数。 default: 3

itag：任务标记值，此标记会在抓取时对比，如果这个值发生改变，不管有效期有没有到都会重新抓取新内容。多数用来动态判断内容是否修改或强制重爬，默认值是：None。

```
def index_page(self, response):
    for item in response.doc('.item').items():
        self.crawl(item.find('a').attr.url, callback=self.detail_page,       itag=item.find('.update-time').text())
```

解析：使用页面中 update-time 元素的值当成 itag 来判断内容是否有更新。

```
class Handler(BaseHandler):
    crawl_config = { 'itag': 'v223' }
```

解析：修改全局参数 itag，使所有任务都重新执行（需要点 run 按钮来启动任务）。

auto_recrawl：当启用,任务将会重新抓取每个时代。默认值：False

```
def on_start(self):
    self.crawl('http://www.example.org/', callback=self.callback,age=5*60*60, auto_recrawl=True)
```

解析：页面每 5 个小时重新启动

method：HTTP 请求方法设置，默认值: GET

params：把一个字典参数附加到 url 参数里，如 ：

```
def on_start(self):
    self.crawl('http://httpbin.org/get', callback=self.callback,params={'a': 123, 'b': 'c'})
    self.crawl('http://httpbin.org/get?a=123&b=c', callback=self.callback)
```

解析：这两个是相同的任务.

data：这个参数会附加到ＵＲＬ请求的 body 里，如果是字典会经过 form-encoding 编码再附加。

```
def on_start(self):
    self.crawl('http://httpbin.org/post', callback=self.callback,method='POST', data={'a': 123, 'b': 'c'})
```

files：dictionary of {field: {filename: 'content'}} files to multipart upload。`

headers：自定义请求头（字典类型）.

cookies：自定义请求的 cookies（字典类型）.

connect_timeout：指定请求时链接超时时间,单位秒，默认值：20。

timeout：请求内容里最大等待秒数．默认值：120.

allow_redirects：遇到 30x 状态码时是否重新请求跟随．默认是：True。

validate_cert：遇到 HTTPS 类型的 URL 时是否验证证书，默认值：True。

proxy：设置代理服务器，格式如 username:password@hostname:port 。暂时只支持 http 代理

```
class Handler(BaseHandler):
    crawl_config = { 'proxy': 'localhost:8080' }
```

解析：Handler.crawl_config 里配置 proxy 参数会对整个项目生效，本项目的所有任务都会使用代理爬取.

etag：如果页面内容没有更改，则使用 HTTP Etag 机制来传递该进程。默认值: True

last_modified：use HTTP Last-Modified header mechanism to pass the process if the content of the page is not changed。 default: True

fetch_type：设置是否启用 JavaScript 解析引擎. default: None

js_script：JavaScript run before or after page loaded, should been wrapped by a function like function() { document.write("binux"); }.

```
def on_start(self):
    self.crawl('http://www.example.org/', callback=self.callback,fetch_type='js', js_script='''
               function() {
                   window.scrollTo(0,document.body.scrollHeight);
                   return 123;
               }
               ''')
```

解析：该脚本将页面滚动到底部。函数的返回值可以通过 Response.js_script_result 捕获

js_run_at：run JavaScript specified via js_script at document-start or document-end. default: document-end

js_viewport_width/js_viewport_height：set the size of the viewport for the JavaScript fetcher of the layout process.

load_images：load images when JavaScript fetcher enabled. default: False

save：传递一个对象给任务，在任务解析时可以通过 response.save 来获取传递的值.

```
def on_start(self):
    self.crawl('http://www.example.org/', callback=self.callback,save={'a': 123}) def callback(self, response): return
 response.save['a']
```

解析：在回调里 123 将被返回.

taskid：唯一性的 taskid 用来区别不同的任务. 默认 taskid 是由 URL 经过 md5 计算得出. 你也可以使用 def get_taskid(self, task)方法覆盖系统自带的来自定义任务 id.如：

```
import json from pyspider.libs.utils
import md5string
def get_taskid(self, task):
    return md5string(task['url']+json.dumps(task['fetch'].get('data', '')))
```

解析：本实例任务 ID 不只是 url，不同的 data 参数也会生成不同的任务 id

force_update：force update task params even if the task is in ACTIVE status.

cancel：cancel a task, should be used with force_update to cancel a active task. To cancel an auto_recrawltask, you should set auto_recrawl=False as well.

2.6.2 Response

Response 对象的方法及成员参考

Response.url：返回最后的 URL 地址

Response.text：请求响应的文本格式内容

如果 Response.encoding 是 None 或 chardet 模块可用，响应内容会自动被解析为指定的编码。

Response.content：请求响应的二进制格式内容，未做编码解析

Response.doc：本方法会调用 PyQuery 库用返回的内容生成一个 PyQuery 对象以方便使用，生成对象时默认已经把里面的所有链接格式化成绝对链接，可直接分析使用.

Response.etree：本方法会调用 lxml 库用返回的内容生成一个 lxml 对象以方便使用.

Response.json：本方法会调用 JSON 相关库来解析返回的内容

Response.headers：请求响应的头信息，dict 格式

Response.error：fetch 的报错信息

Response.time：抓取使用的时间

Response.ok：如果状态码是 200 并且没有错误信息这个值就是 True，用来判断是否请求成功。

Response.save：The object saved by self.crawl API

Response.js_script_result：content returned by JS script

Response.raise_for_status()：Raise HTTPError if status code is not 200 or Response.error exists

2.6.3 @every(minutes=0, seconds=0)

设置多少分钟或秒运行一次此方法,常用来定时执行任务.

```
@every(minutes=24 * 60)
def on_start(self): for url in urllist:
    self.crawl(url, callback=self.index_page)
```

如上实例是这个 URL 每 24 小时被执行一次，请注意:如果 age 设置的时间还没过期或有 period 设置的比较长,这个 url 是不会被重新运行抓取。

```
@every(minutes=24 * 60)
def on_start(self):
    self.crawl('http://www.example.org/',callback=self.index_page)
@config(age=10 * 24 * 60 * 60)
    def index_page(self):
        ...
```

本实例设置的是每天执行一次，但是过期时间设置的是十天才过期，所以即使每天都调用 on_start 也不会重每天都执行抓取，十天内只抓一次。

三、入门 demo

以豆瓣电影为例：http://movie.douban.com/

3.1 选取一个开始网址

首先我们需要抓一个电影列表，一个好的列表应该：

1). 包含足够多的电影的 URL

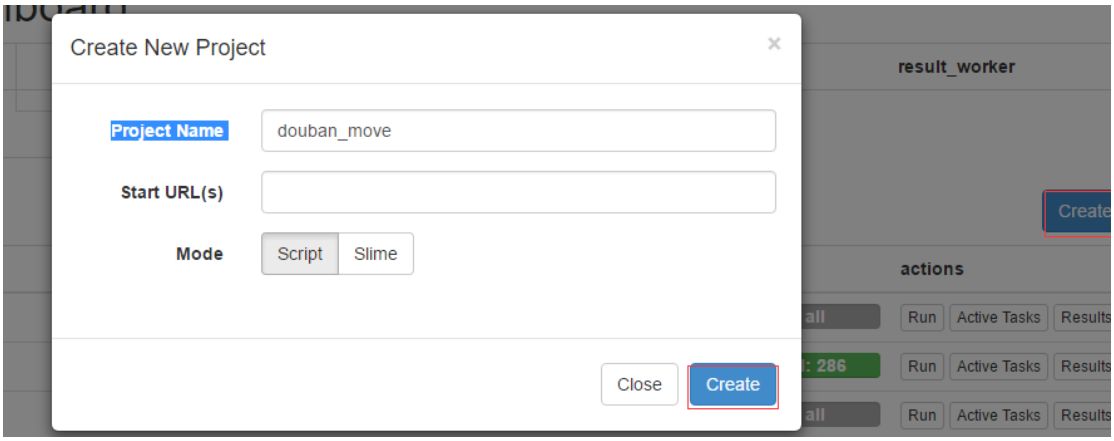
2). 通过翻页，可以遍历到所有的电影

3). 一个按照更新时间排序的列表，可以更快抓到最新更新的电影

在 http://movie.douban.com 中，发现并没有一个列表能包含所有电影，只能退而求其次，通过抓取分类下的所有的标签列表页，来遍历所有的电影：http://movie.douban.com/tag/

3.2 创建一个项目

点击 create

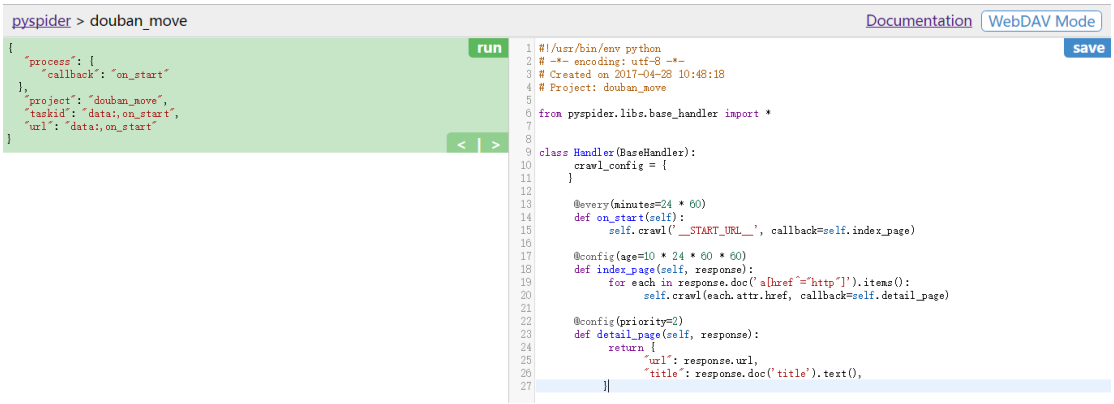


Project Name：项目名称 必填

Start URL(s)：起始网址 选填

Mode：

点 create 后打开代码编辑器（代码编辑器默认有简单的实例代码），如下图



右侧就是代码编辑器，以后可以直接在这添加修改代码

代码简单分析：

1). def on_start(self) 方法是入口代码。当在 web 控制台点击 run 按钮时会执行此方法。

2). self.crawl(url, callback=self.index_page)这个方法是调用 API 生成一个新的爬取任务，这个任务被添加到待抓取队列。

3). def index_page(self, response) 这个方法获取一个 Response 对象。 response.doc 是 pyquery 对象的一个扩展方法。pyquery 是一个类似于 jQuery 的对象选择器。

4). def detail_page(self, response)返回一个结果集对象。这个结果默认会被添加到 resultdb 数据库（如果启动时没有指定数据库默认调用 sqlite 数据库）。你也可以重写 on_result(self,result)方法来指定保存位置。

5). @every(minutes=24*60, seconds=0) 这个设置是告诉 scheduler（调度器）on_start 方法每天执行一次。

6). @config(age=10 * 24 * 60 * 60) 这个设置告诉 scheduler（调度器）这个 request（请求）过期时间是 10 天，10 天内再遇到这个请求直接忽略。此参数也可以在 self.crawl(url, age=10*24*60*60) 和 crawl_config 中设置。

7). @config(priority=2) 这个是优先级设置,数字越小越先执行。

3.3 调试脚本

3.3.1 开始

替换 on_start 函数的 self.crawl 的 URL：

```
@every(minutes=24 * 60)
def on_start(self):
    self.crawl('http://movie.douban.com/tag/', callback=self.index_page)
```

self.crawl 告诉 pyspider 抓取指定页面，然后使用 callback 函数对结果进行解析。
@every 修饰器，表示 on_start 每天会执行一次
点击绿色的 run 执行，切换到 follows 面板，点击绿色的播放按钮：

web：在 web 控制台以浏览器模式显示页面内容
hTML ：在回调函数时显示ＨＴＭＬ代码
follows：查看可以从当前回调中获得的回调
messages ：显示来自 self.send_message API 接口的消息
enable CSS Selector Helper：css 选择器助手可以在 web 视图中通过单击获取节点的 css 规则，并自动插件到脚本。

3.3.2 Tag 列表页

在 tag 列表页中，我们需要提取出所有的 电影列表页 的 URL。
由于电影列表页和 tag 列表页长的并不一样，在这里新建了一个 callback 为 self.list_page。
再次点击 run 让我们进入一个电影列表页(list_page)。在这个页面中我们需要提取：电影的链接，例如，http://movie.douban.com/subject/1292052/以及下一页的链接，用来翻页。

3.3.3 获取方法

此处介绍三种方法：css 选择器、CSS Selector Helper、Chrome Dev Tools

1、Chrome Dev Tools

可以在 Chrome Dev Tools 的帮助下，找到合适的表达式，先按键盘 F12，具体操作如下图

2、CSS 选择器

CSS 选择器，是 CSS 用来定位需要设置样式的元素所使用的表达式，可以通过它定位需要的元素。例如：获取豆瓣电影标签中类型分类，点击 html 页面，

```
<table class="tagCol">
  <tbody>
    <tr>
      <td>
        <a href="/tag/爱情">
          爱情
        </a>
        <b>
          (11463499)
        </b>
      </td>
      <td>
        <a href="/tag/喜剧">
          喜剧
        </a>
        <b>
          (9494033)
        </b>
      </td>
      <td>
        <a href="/tag/剧情">
          剧情
        </a>
        <b>
```

enable css selector helper | web | html | follow 136 | messages

```
def index_page(self, response):
    for each in response.doc('.tagCol>tbody>tr>td>a').items():
        self.crawl(each.attr.href, callback=self.list_page)
```

可参照《CSS 选择器参考手册》，http://www.w3school.com.cn/cssref/css_selectors.asp。

3、CSS Selector Helper

在 pyspider 中，内置了一个 CSS Selector Helper，切换到 web 页面，点击 Enable CSS selector helper 按钮

开启后，鼠标放在元素上，会被黄色高亮，点击后，所有拥有相同 CSS 选择器 表达式

的元素会被高亮。点击箭头位置 copy css selector 表达式会被复制，然后点击 add to editor 插入到代码当前光标位置。创建下面的代码，将光标停留在单引号中间：

```
def list_page(self, response):
    for each in response.doc('').items():
```

点击一个电影的链接，CSS 选择器表达式将会插入代码中，

```
def list_page(self, response):
    for each in response.doc('td > .pl2 > a').items():
        self.crawl(each.attr.href, priority=9, callback=self.detail_page)
```

如此重复，插入翻页的链接：

```
def list_page(self, response):
    for each in response.doc('td > .pl2 > a').items():
        self.crawl(each.attr.href, priority=9, callback=self.detail_page)
    for next in response.doc('.next > a').items():
        self.crawl(next.attr.href, callback=self.list_page)
```

翻页是一个到自己的 callback 回调

3.4.4 电影详情页

再次点击 run，follow 到详情页，分别添加电影标题，打分和导演等：

```
@config(priority=3)
def detail_page(self, response):
    return {
        "url": response.url,
        "title": response.doc('h1 > span').text(),
        "rating": response.doc('.rating_num').text(),
        "DERRECTOR": response.doc('#info>span:first-child>span>a:first-
child').text(), #导演

"PRODUCE_YEAR":response.doc('span[property*="initialReleaseDate"]').text(),
        # 出品年份
```

3.4 执行任务

当完成脚本编写，调试无误后，请先保存脚本！然后返回到控制台首页。以下为控制台使用说明：

队列统计是为了方便查看爬虫状态，优化爬虫爬取速度新增的状态统计。每个组件之间的数字就是对应不同队列的排队数量。通常来是 0 或是个位数，如果达到了几十甚至一百说明下游组件出现了瓶颈或错误，需要分析处理。

新建项目：pyspider 与 scrapy 最大的区别就在这，pyspider 新建项目调试项目完全在 web 下进行，而 scrapy 是在命令行下开发并运行测试.

组名：项目新建后一般来说是不能修改项目名的，如果需要特殊标记可修改组名．直接在组名上点鼠标左键进行修改。注意：group 命名中不能含有中文字符，组名改为 delete 后如果状态为 stop 状态，24 小时后项目会被系统删除。

运行状态：这一栏显示的是当前项目的运行状态。每个项目的运行状态都是单独设置的，直接在每个项目的运行状态上点鼠标左键进行修改。运行分为五个状态：TODO、STOP、CHECKING、DEBUG、RUNNING。各状态说明：

TODO 是新建项目后的默认状态，不会运行项目

STOP 状态是停止状态，也不会运行

CHECHING 是修改项目代码后自动变的状态

DEBUG 是调试模式，遇到错误信息会停止继续运行

RUNNING 是运行状态，遇到错误会自动尝试，如果还是错误会跳过错误的任务继续运行.

速度控制：这个功能是速度设置项，rate 是每秒爬取页面数，burst 是并发数．如 1/3 是三个并发，每秒爬取一个页面。

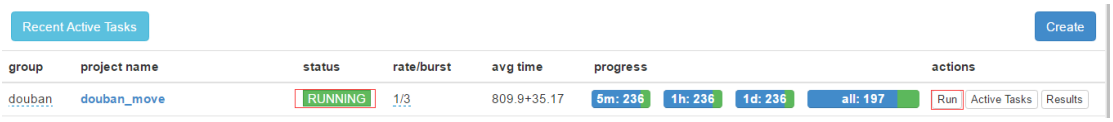
简单统计：这个功能只是简单的做的运行状态统计，5m 是五分钟内任务执行情况，1h 是一小时内运行任务统计，1d 是一天内运行统计，all 是所有的任务统计。

运行：run 按钮是项目初次运行需要点的按钮，这个功能会运行项目的 on_start 方法来生成入口任务。

任务列表：显示最新任务列表，方便查看状态，查看错误等。

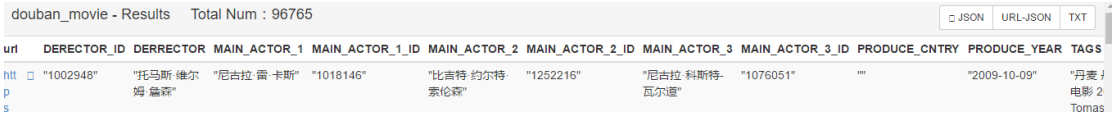
结果查看：查看项目爬取的结果

将状态由 TODO 改成 debug 或 running，然后 RUN 按钮启动项目。



3.5 查看结果

点击 Active Tasks 可查看进程，Results 查看和下载爬取结果，如下图



Total Num：结果总数

json、url_json、txt 三种结果格式

3.6 删除项目

把 group 设置成 delete 并把项目状态设置成 STOP，24 小时后系统会自动删除此项目。