目錄

PHPSpider开发文档

PHP蜘蛛爬虫开发文档

《我用爬虫一天时间“偷了”知乎一百万用户，只为证明PHP是世界上最好的语言》所使用的程序框架

编写PHP网络爬虫, 需要具备以下技能:

- 爬虫采用PHP编写
- 从网页中抽取数据需要用XPath (XPath选择器教程)
- 当然我们还可以使用CSS选择器 (CSS选择器教程)
- 很多情况下都会用到正则表达式 (正则表达式教程)
- Chrome的开发者工具是神器, 很多AJAX请求需要用它来分析

注意：本框架只能在命令行下运行，命令行、命令行、命令行，重要的事情说三遍 ^_^

第一个demo

爬虫采用PHP编写, 下面以糗事百科为例, 来看一下我们的爬虫长什么样子:

安装

1、通过GitHub下载

```
require_once __DIR__ . '/../autoloader.php';
use phpspider\core\phpspider;
```

2、通过composer下载

```
composer require owner888/phpspider
```

```
require './vendor/autoload.php';
use phpspider\core\phpspider;
```

3、加上一段很讨厌的注释，别问我为什么，我就是这么讨厌 ^_^

```
/* Do NOT delete this comment */
/* 不要删除这段注释 */
```

```
$configs = array(
    'name' => '糗事百科',
    'domains' => array(
        'qiushibaike.com',
        'www.qiushibaike.com'
    ),
    'scan_urls' => array(
        'http://www.qiushibaike.com/'
    ),
    'content_url_regexes' => array(
        "http://www.qiushibaike.com/article/\d+"
    ),
    'list_url_regexes' => array(
        "http://www.qiushibaike.com/8hr/page/\d+\?s=\d+"
    ),
    'fields' => array(
        array(
            // 抽取内容页的文章内容
            'name' => "article_content",
            'selector' => "//*[@id='single-next-link']",
            'required' => true
        ),
        array(
            // 抽取内容页的文章作者
            'name' => "article_author",
            'selector' => "//div[contains(@class,'author')]//h2"
,
            'required' => true
        ),
    ),
);
$spider = new phpspider($configs);
$spider->start();
```

爬虫的整体框架就是这样,首先定义了一个$configs数组,里面设置了待爬网站的一些信息,然后通过调用 `$spider = new phpspider($configs);` 和 `$spider->start();` 来配置并启动爬虫.

运行界面如下:

$configs对象如何定义, 后面会作详细介绍.^_^

configs详解——之成员

爬虫的整体框架是这样：首先定义了一个$configs数组, 里面设置了待爬网站的一些信息, 然后通过调用 `$spider = new phpspider($configs);` 和 `$spider->start();` 来配置并启动爬虫.

$configs数组中可以定义下面这些元素

name

> 定义当前爬虫名称

String类型 可选设置

举个栗子:

```
'name' => '糗事百科'
```

log_show

> 是否显示日志
> 为true时显示调试信息
> 为false时显示爬取面板

布尔类型 可选设置

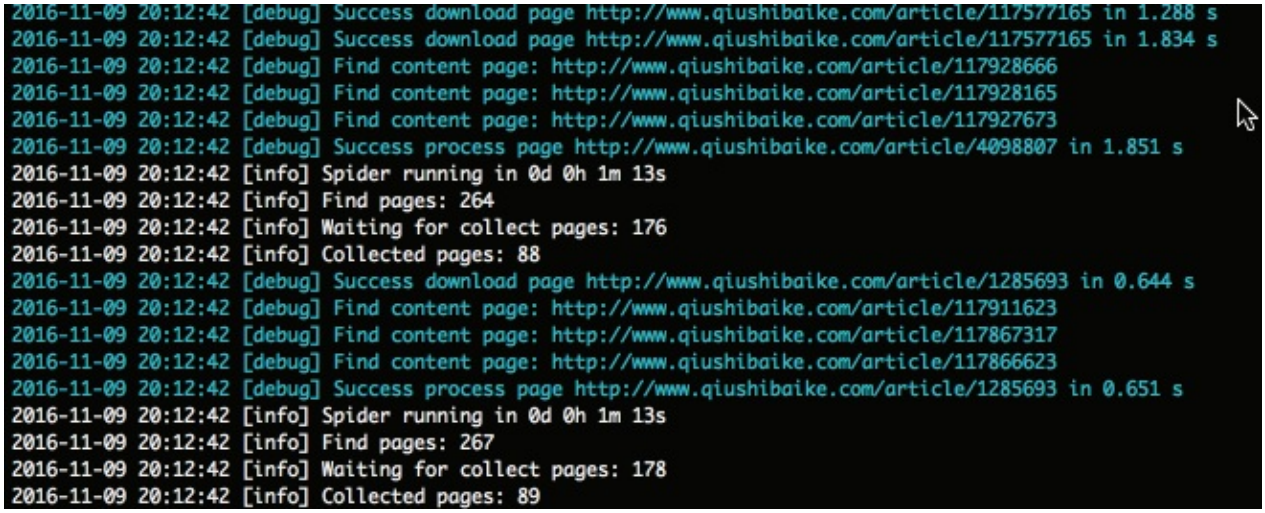
log_show默认值为**false**，即显示爬取面板

栗子1:

```
'log_show' => false
```

栗子2:

```
'log_show' => true
```



注意：显示爬取面板时，也可以通过**tail**命令来查看日志

```
tail -f data/phpspider.log
```

log_file

> 日志文件路径

String类型 可选设置

log_file默认路径为**data/phpspider.log**

举个栗子:

```
'log_file' => data/qiushibaike.log
```

log_type

> 显示和记录的日志类型
> 普通类型: info
> 警告类型: warn
> 调试类型: debug
> 错误类型: error

String类型 可选设置

log_type默认值为空，即显示和记录所有日志类型

栗子1:
显示错误日志

```
'log_type' => 'error'
```

举个栗子:
显示错误和调试日志

```
'log_type' => 'error,debug'
```

input_encoding

> 输入编码
> 明确指定输入的页面编码格式(UTF-8,GB2312,.....)，防止出现乱码,如果设置null则自动识别

String类型 可选设置

input_encoding默认值为**null**，即程序自动识别页面编码

举个栗子:

```
'input_encoding' => 'GB2312'
```

output_encoding

> 输出编码
> 明确指定输出的编码格式(UTF-8,GB2312,.....)，防止出现乱码,如果设置null则为utf-8

String类型 可选设置

output_encoding默认值为**utf-8,** 如果数据库为**gbk**编码，请修改为**gb2312**

举个栗子:

```
'output_encoding' => 'GB2312'
```

tasknum

> 同时工作的爬虫任务数
> 需要配合redis保存采集任务数据，供进程间共享使用

整型 可选设置

tasknum默认值为**1**，即单进程任务爬取

举个栗子:

开启5个进程爬取网页

```
'tasknum' => 5
```

> `tasknum` 在如何实现多任务爬虫中作详细介绍。

multiserver

> 多服务器处理
> 需要配合redis来保存采集任务数据，供多服务器共享数据使用

布尔类型 可选设置

multiserver默认值为**false**

举个栗子:

```
'multiserver' => true
```

> `multiserver` 在如何实现多服务器集群爬虫中作详细介绍。

serverid

> 服务器ID

整型 可选设置

serverid默认值为**1**

举个栗子:
启用第二台服务器

```
'serverid' => 2
```

save_running_state

> 保存爬虫运行状态
> 需要配合redis来保存采集任务数据，供程序下次执行使用
> 注意：多任务处理和多服务器处理都会默认采用redis，可以不设置这个参数

布尔类型 可选设置

save_running_state默认值为**false**，即不保存爬虫运行状态

举个栗子:

```
'save_running_state' => true
```

queue_config

> redis配置

数组类型 可选设置

保存爬虫运行状态、多任务处理 和 多服务器处理 都需要 `redis` 来保存采集任务数据

举个栗子:

```
'queue_config' => array(
    'host'      => '127.0.0.1',
    'port'      => 6379,
    'pass'      => '',
    'db'        => 5,
    'prefix'    => 'phpspider',
    'timeout'   => 30,
)
```

proxy

> 代理服务器
> 如果爬取的网站根据IP做了反爬虫, 可以设置此项

数组类型 可选设置

栗子1:
普通代理

```
'proxy' => array('http://host:port')
```

栗子2:
验证代理

```
'proxy' => array('http://user:pass@host:port')
```

注意：如果对方根据**IP**做了反爬虫技术，你可能需要到 阿布云代理 申请代理通道或者 第三方免费代理**IP**，然后在这里填写代理信息

interval

> 爬虫爬取每个网页的时间间隔
> 单位：毫秒

整型 可选设置

举个栗子:
设置爬取时间间隔为1秒

```
'interval' => 1000
```

timeout

> 爬虫爬取每个网页的超时时间
> 单位：秒

整型 可选设置

timeout默认值为**5**秒

举个栗子:

```
'timeout' => 5
```

max_try

> 爬虫爬取每个网页失败后尝试次数
> 网络不好可能导致爬虫在超时时间内抓取失败, 可以设置此项允许爬虫重复爬取

整型 可选设置

max_try默认值为**0**，即不重复爬取

举个栗子:

```
'max_try' => 5 // 重复爬取5次
```

max_depth

> 爬虫爬取网页深度，超过深度的页面不再采集
> 对于抓取最新内容的增量更新，抓取好友的好友的好友这类型特别有用

整型 可选设置

`max_depth` 默认值为**0**，即不限制

举个栗子:
采集知乎好友时只采集到5级深度

```
'max_depth' => 5
```

max_fields

> 爬虫爬取内容网页最大条数
> 抓取到一定的字段后退出

整型 可选设置

`max_fields` 默认值为**0**，即不限制

举个栗子:

```
'max_fields' => 100
```

user_agent

> 爬虫爬取网页所使用的浏览器类型
> `phpspider::AGENT_ANDROID`
> `phpspider::AGENT_IOS`
> `phpspider::AGENT_PC`
> `phpspider::AGENT_MOBILE`

枚举类型 可选设置

栗子1:
使用内置的枚举类型

`phpspider::AGENT_ANDROID`,表示爬虫爬取网页时,使用安卓手机浏览器
`phpspider::AGENT_IOS`,表示爬虫爬取网页时,使用苹果手机浏览器
`phpspider::AGENT_PC`,表示爬虫爬取网页时,使用**PC**浏览器
`phpspider::AGENT_MOBILE`,表示爬虫爬取网页时,使用移动设备浏览器

```
'user_agent' => phpspider::AGENT_ANDROID
```

栗子2:
使用自定义类型

```
'user_agent' => "Mozilla/5.0"
```

栗子3:

随机浏览器类型，用于破解防采集

```
'user_agent' => array(
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit
/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36",
    "Mozilla/5.0 (iPhone; CPU iPhone OS 9_3_3 like Mac OS X) App
leWebKit/601.1.46 (KHTML, like Gecko) Version/9.0 Mobile/13G34 S
afari/601.1",
    "Mozilla/5.0 (Linux; U; Android 6.0.1;zh_cn; Le X820 Build/F
EXCNFN5801507014S) AppleWebKit/537.36 (KHTML, like Gecko)Version
/4.0 Chrome/49.0.0.0 Mobile Safari/537.36 EUI Browser/5.8.015S",
)
```

client_ip

> 爬虫爬取网页所使用的伪IP，用于破解防采集

String类型 或 数组类型 可选设置

栗子1:

```
'client_ip' => '192.168.0.2'
```

栗子2:

随机伪造IP，用于破解防采集

```
'client_ip' => array(
    '192.168.0.2',
    '192.168.0.3',
    '192.168.0.4',
)
```

export

> 爬虫爬取数据导出
>
> > type：导出类型 csv、sql、db
> > file：导出 csv、sql 文件地址
> > table：导出db、sql数据表名

注意导出到数据库的表和字段要和上面的**fields**对应

数组类型 可选设置

栗子1:
导出CSV结构数据到文件

```
'export' => array(
    'type' => 'csv',
    'file' => './data/qiushibaike.csv', // data目录下
)
```

栗子2:
导出SQL语句到文件

```
'export' => array(
    'type'  => 'sql',
    'file'  => './data/qiushibaike.sql',
    'table' => '数据表',
)
```

栗子3:
导出数据到Mysql

```
'export' => array(
    'type' => 'db',
    'table' => '数据表',  // 如果数据表没有数据新增请检查表结构和字段名是否匹配
)
```

db_config

数据库配置

```
'db_config' => array(
    'host'  => '127.0.0.1',
    'port'  => 3306,
    'user'  => 'root',
    'pass'  => 'root',
    'name'  => 'demo',
)
```

domains

> 定义爬虫爬取哪些域名下的网页, 非域名下的url会被忽略以提高爬取速度

数组类型 不能为空

举个栗子:

```
'domains' => array(
    'qiushibaike.com',
    'www.qiushibaike.com'
)
```

scan_urls

> 定义爬虫的入口链接,爬虫从这些链接开始爬取,同时这些链接也是监控爬虫所要监控的链接

数组类型 不能为空

举个栗子:

```
'scan_urls' => array(
    'http://www.qiushibaike.com/'
)
```

content_url_regexes

> 定义内容页url的规则
> 内容页是指包含要爬取内容的网页 比如 `http://www.qiushibaike.com/article/115878724` 就是糗事百科的一个内容页

数组类型 正则表达式 最好填写以提高爬取效率

举个栗子:

```
'content_url_regexes' => array(
    "http://www.qiushibaike.com/article/\d+",
)
```

list_url_regexes

> 定义列表页url的规则
> 对于有列表页的网站,使用此配置可以大幅提高爬虫的爬取速率
> 列表页是指包含内容页列表的网页 比如 `http://www.qiushibaike.com/8hr/page/2/?s=4867046` 就是糗事百科的一个列表页

数组类型 正则表达式

举个栗子:

```
'list_url_regexes' => array(
    "http://www.qiushibaike.com/8hr/page/\d+\?s=\d+"
)
```

fields

> 定义内容页的抽取规则
> 规则由一个个 `field` 组成, 一个 `field` 代表一个数据抽取项

数组类型 不能为空

举个栗子:

```
'fields' => array(
    array(
        'name' => "content",
        'selector' => "//*[@id='single-next-link']",
        'required' => true,
    ),
    array(
        'name' => "author",
        'selector' => "//div[contains(@class,'author')]//h2",
    )
)
```

上面的例子从网页中抽取内容和作者, 抽取规则是针对糗事百科的内容页写的

> `field` 在configs详解——之field中作详细介绍。

configs详解——之field

`field` 定义一个抽取项, 一个 `field` 可以定义下面这些东西

name

> 给此项数据起个变量名
> 变量名中不能包含.
> 如果抓取到的数据想要以文章或者问答的形式发布到网站(WeCenter, WordPress, Discuz!等), field的命名请参考两个完整demo中的命名, 否则无法发布成功

String类型 不能为空

举个栗子:

给 `field` 起了个名字叫 `content`

```
array(
    'name' => "content",
    'selector' => "//*[@id='single-next-link']"
)
```

selector

> 定义抽取规则, 默认使用xpath
> 如果使用其他类型的, 需要指定selector_type

String类型 不能为空

举个栗子:
使用xpath来抽取糗事百科的笑话内容，selector的值就是内容的xpath

```
array(
    'name' => "content",
    'selector' => "//*[@id='single-next-link']"
)
```

selector_type

抽取规则的类型

> 目前可用xpath, jsonpath, regex
> 默认 `xpath`

枚举类型

栗子1:
selector默认使用xpath

```
array(
    'name' => "content",
    'selector' => "//*[@id='single-next-link']" // xpath抽取规则
)
```

栗子2:
使用正则表达式来抽取数据

```
array(
    'name' => "content",
    'selector_type' => 'regex',
    'selector' => '#<div\sclass="content">([^/]+)</div>#i' // re
gex抽取规则
)
```

required

> 定义该 `field` 的值是否必须, 默认false
> 赋值为true的话, 如果该 `field` 没有抽取到内容, 该field对应的整条数据都将被丢弃

布尔类型

举个栗子:

```
array(
    'name' => "content",
    'selector' => "//*[@id='single-next-link']",
    'required' => true
)
```

repeated

> 定义该 `field` 抽取到的内容是否是有多项,默认 `false`
> 赋值为true的话,无论该 `field` 是否眞的是有多项,抽取到的结果都是数组结构

布尔类型

举个栗子:
爬取的网页中包含多条评论，所以抽取评论的时候要将repeated赋值为true

```
array(
    'name' => "comments",
    'selector' => "//*[@id='zh-single-question-page']//a[contains(@class,'zm-item-tag')]",
    'repeated' => true
)
```

children

> 为此 `field` 定义子项
> 子项的定义仍然是一个 `fields` 数组
> 没错,这是一个树形结构

数组类型

举个栗子:
抓取糗事百科的评论，每个评论爬取了内容，点赞数

```
array(
    'name' => "article_comments",
    'selector' => "//div[contains(@class,'comments-wrap')]",
    'children' => array(
        array(
            'name' => "replay",
            'selector' => "//div[contains(@class,'replay')]",
            'repeated' => true,
        ),
        array(
            'name' => "report",
            'selector' => "//div[contains(@class,'report')]",
            'repeated' => true,
        )
    )
)
```

source_type

该field的数据源, 默认从当前的网页中抽取数据
选择 `attached_url` 可以发起一个新的请求, 然后从请求返回的数据中抽取
选择 `url_context` 可以从当前网页的url附加数据（点此查看“url附加数据”实例解析）中抽取

枚举类型

attached_url

当source_type设置为 `attached_url` 时, 定义新请求的url

String类型

举个栗子:
当爬取的网页中某些内容需要异步加载请求时，就需要使用attached_url，比如，抓取知乎回答中的评论部分，就是通过AJAX异步请求的数据

```
array(
    'name' => "comment_id",
    'selector' => "//div/@data-aid",
),
array(
    'name' => "comments",
    'source_type' => 'attached_url',
    // "comments"是从发送"attached_url"这个异步请求返回的数据中抽取的
    // "attachedUrl"支持引用上下文中的抓取到的"field", 这里就引用了上面
抓取的"comment_id"
    'attached_url' => "https://www.zhihu.com/r/answers/{comment_
id}/comments",
    'selector_type' => 'jsonpath'
    'selector' => "$.data",
    'repeated => true,
    'children' => array(
        ...
    )
}
```

configs详解——之requests

`requests` 表示当前正在爬取的网站的对象，下面介绍了可以调用的函数

requests成员

input_encoding

> 输入编码
> 明确指定输入的页面编码格式(UTF-8,GB2312,.....)，防止出现乱码,如果设置null则自动识别

String类型 可选设置

input_encoding默认值为**null**，即程序自动识别页面编码

举个栗子:

```
requests::$input_encoding = 'GB2312';
```

output_encoding

> 输出编码
> 明确指定输出的编码格式(UTF-8,GB2312,.....)，防止出现乱码,如果设置null则为utf-8

String类型 可选设置

output_encoding默认值为**utf-8,** 如果数据库为**gbk**编码，请修改为**gb2312**

举个栗子:

```
requests::$output_encoding = 'GB2312';
```

requests方法

set_timeout($timeout)

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 设置请求超时时间

> @param $timeout 需添加的timeout

默认值为**5**，即**5**秒超时

栗子1:
传入单一的值作为timeout，会同时设置connect和read

```
$spider->on_start = function($phpspider)
{
    requests::set_timeout(10);
};
```

栗子2：
传入数组，会分别设置connect和read二者的timeout

```
requests::set_timeout( array(3, 27) );
```

栗子3：
如果远端服务器很慢，你可以让requests永远等待，传入一个 0 作为 timeout 值，然后就冲咖啡去吧。

```
requests::set_timeout(0);
```

set_proxy($proxy)

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 设置请求代理

> @param $proxy 需添加的代理，用于破解防采集，支持字符串和数组类型传入

栗子1：

字符串类型

```
$spider->on_start = function($phpspider)
{
    requests::set_proxy('http://user:pass@host:port);
};
```

栗子2:

数组类型，代理有多个

```
$spider->on_start = function($phpspider)
{
    // 代理如果有多个，请求时会随机采用
    $proxy = array(
        'http://user1:pass1@host:port',
        'http://user2:pass2@host:port',
    );
    requests::set_proxy($proxy);
};
```

set_useragent($useragent)

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 设置浏览器useragent

> @param $useragent 需添加的useragent

默认使用**useragent: requests/2.0.0**

点击查看“常见浏览器useragent大全”

栗子1:

```
$spider->on_start = function($phpspider)
{
    requests::set_useragent("Mozilla/4.0 (compatible; MSIE 6.0;
) Opera/UCWEB7.0.2.37/28/");
};
```

栗子2:

随机伪造useragent，传递数组即可，爬虫在请求的时候就会随机取一个useragent访问对方网站，让对方网站对useragent的反爬虫限制失

```
$spider->on_start = function($phpspider)
{
    requests::set_useragent(array(
        "Mozilla/4.0 (compatible; MSIE 6.0; ) Opera/UCWEB7.0.2.3
7/28/",
        "Opera/9.80 (Android 3.2.1; Linux; Opera Tablet/ADR-1109
081720; U; ja) Presto/2.8.149 Version/11.10",
        "Mozilla/5.0 (Android; Linux armv7l; rv:9.0) Gecko/20111
216 Firefox/9.0 Fennec/9.0"
    ));
};
```

set_referer($referer)

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 设置请求来路URL

> @param $referer 需添加的来路URL，用于破解防采集

举个栗子:

```
$spider->on_start = function($phpspider)
{
    requests::set_referer("http://www.qiushibaike.com");
};
```

set_header($key, $value)

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 用来添加一些HTTP请求的Header

> @param $key Header的key, 如User-Agent,Referer等
> @param $value Header的值

举个栗子:
Referer是HTTP请求Header的一个属性，`http://www.9game.cn/kc/`是Referer的值

```
$spider->on_start = function($phpspider)
{
    requests::set_header("Referer", "http://www.9game.cn/kc/");
};
```

set_cookie($key, $value, $domain='')

一般在`on_start`回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 用来添加一些HTTP请求的Cookie

> @param $key Cookie的key
> @param $value Cookie的值
> @param $domain 默认放到全局Cookie，设置域名后则放到相应域名下

举个栗子:
cookie是由键-值对组成的，BAIDUID是cookie的key，FEE96299191CB0F11954F3A0060FB470:FG=1则是cookie的值

```
$spider->on_start = function($phpspider)
{
    requests::set_cookie("BAIDUID", "FEE96299191CB0F11954F3A0060
FB470:FG=1");
    // 把Cookie设置到 www.phpspider.org 域名下
    requests::set_cookie("NAME", "phpspider", "www.phpspider.org
");
};
```

get_cookie($name, $domain = '')

一般在`on_start`回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 用来得到某个域名所附带的某个Cookie

> @param $name Cookie的名称
> @param $domain configs的domains成员中的元素

举个栗子:
得到 `s.weibo.com` 域名所附带的 `Cookie` , 并将 `Cookie` 添加到 `weibo.com` 的域名中

```
$configs = array(
    'domains' => array(
        's.weibo.com',
        'weibo.com'
    )
    // configs的其他成员
    ...
);

$spider->on_start = function($phpspider)
{
    $cookie = requests::get_cookie("SUB", "s.weibo.com");
    // 把Cookie设置到 weibo.com 域名下
    requests::set_cookie("SUB", $cookie, "weibo.com");
};
```

set_cookies($cookies, $domain='')

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 用来添加一些HTTP请求的Cookie

> @param $cookies 多个Cookie组成的字符串
> @param $domain 默认放到全局Cookie，设置域名后则放到相应域名下

举个栗子:
cookies是多个cookie的键-值对组成的字符串，用;分隔。BAIDUID和BIDUPSID是cookie的key，FEE96299191CB0F11954F3A0060FB470:FG=1和FEE96299191CB0F11954F3A0060FB470是cookie的值，键-值对用=相连

```
$spider->on_start = function($phpspider)
{
    requests::set_cookies("BAIDUID=FEE96299191CB0F11954F3A0060FB
470:FG=1; BIDUPSID=FEE96299191CB0F11954F3A0060FB470;");
    // 把Cookie设置到 www.phpspider.org 域名下
    requests::set_cookies("NAME", "www.phpspider.org");
};
```

get_cookies($domain = '')

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 用来得到某个域名所附带的所有Cookie

> @param $domain configs的domains成员中的元素
> @return array 返回的是所有Cookie的数组

举个栗子:
得到 `s.weibo.com` 域名所附带的 `Cookie` , 并将 `Cookie` 添加到 `weibo.com` 的域名中

```
$configs = array(
    'domains' => array(
        's.weibo.com',
        'weibo.com'
    )
    // configs的其他成员
    ...
);

$spider->on_start = function($phpspider)
{
    $cookies = requests::get_cookies("s.weibo.com");
    // 返回的是数组，可以输出看看所有的Cookie内容
    print_r($cookies);
    // 数组转化成String
    $cookies = implode(";", $cookies);
    // 把Cookie设置到 weibo.com 域名下
    requests::set_cookies($cookies, "weibo.com");

};
```

set_client_ip($ip)

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 设置请求伪IP

> @param $ip 需添加的伪IP

栗子1:

```
$spider->on_start = function($phpspider)
{
    requests::set_client_ip("192.168.0.2");
};
```

栗子2:

随机伪造IP，只需要添加数组即可

```
$spider->on_start = function($phpspider)
{
    $ips = array(
        "192.168.0.2",
        "192.168.0.3",
        "192.168.0.4"
    );
    requests::set_client_ip($ips);
};
```

set_hosts($host, $ips)

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 设置请求的第三方主机和IP

> @param $hosts 需添加的主机和IP，用于采集第三方不同的服务器

举个栗子:

```
$spider->on_start = function($phpspider)
{
    $host = "www.qiushibaike.com";
    $ips = array(
        "203.195.143.21",
        "203.195.143.22"
    );
    requests::set_hosts($host, $ips);
};
```

get($url, $params, $allow_redirects, $cert)

可以在任何地方调用, 用来获取某个网页

> @param $url 请求URL
> @param $params 请求参数
> @param $allow_redirects 是否允许获取跳转后的页面
> @param $cert 证书

举个栗子:
获取 Github 的公共时间线

```
$json = requests::get("https://github.com/timeline.json");
$data = json_decode($json, true);
print_r($data);
```

post($url, $params, $files, $allow_redirects, $cert)

可以在任何地方调用, 用来获取某个网页

> @param $url 请求URL
> @param $params 请求参数
> @param $files 上传文件
> @param $allow_redirects 是否允许获取跳转后的页面
> @param $cert 证书

栗子1:
用户登录

```
$params = array(
    'username' => 'test888',
    'password' => '123456',
);
$html = requests::post("http://www.domain.com", $params);
```

栗子2：
文件上传

```
$files = array(
    'file1' => 'test.jpg',
    'file2' => 'test.png'
);
$html = requests::post("http://www.domain.com", NULL, $files);
```

服务端代码

```
if ($_FILES["file1"]["error"] > 0)
{
    echo "错误：" . $_FILES["file1"]["error"] . "<br\>";
}
else
{
    echo "上传文件名: " . $_FILES["file1"]["name"] . "<br\>";
    echo "文件类型: " . $_FILES["file1"]["type"] . "<br\>";
    echo "文件大小: " . ($_FILES["file1"]["size"] / 1024) . " kB<
br\>";
    echo "文件临时存储的位置: " . $_FILES["file1"]["tmp_name"];
}
```

put($url, $params, $allow_redirects, $cert)

可以在任何地方调用, 用来获取某个网页

> @param $url 请求URL
> @param $params 请求参数
> @param $allow_redirects 是否允许获取跳转后的页面
> @param $cert 证书

举个栗子:
添加用户 test888

```
$params="{username:\"test888\",username:\"123456\"}";
$html = requests::put("http://www.domain.com", $params);
```

delete($url, $params, $allow_redirects, $cert)

可以在任何地方调用, 用来获取某个网页

> @param $url 请求URL
> @param $params 请求参数
> @param $allow_redirects 是否允许获取跳转后的页面
> @param $cert 证书

举个栗子:
删除用户 test888

```
$params="{username:\"test888\"}";
$html = requests::delete("http://www.domain.com", $params);
```

获取网页编码

可以使用 requests::$encoding 来获取网页编码

```
requests::get("http://www.domain.com");
echo requests::$encoding;
// utf-8
```

当你发送请求时，requests会根据HTTP头部来猜测网页编码，当HTTP头部无法获取时，会继续用网页内容来继续猜测，当你使用 requests::$text 时，requests就会使用这个编码。当然你还可以修改reuqests的编码形式。

```
requests::$output_encoding = 'gbk';
requests::get("http://www.domain.com");
echo requests::$encoding;
// gbk
```

像上面的例子，请求前先指定要输出的编码为gbk，获取到的网页内容就是gbk编码的内容。

json

现在很多PHP环境都默认自带json扩展，并不需要像python、golang那样需要去引入新模块，下面是查询IP的API 例子

```
$json = requests::get('http://ip.taobao.com/service/getIpInfo.ph
p?ip=122.88.60.28');
$rs = json_decode($json, true);
echo $rs['data']['country'];
// 中国
```

获取响应内容

可通过 requests::$content、requests::$text 来获取转码前后网页的内容。

```
requests::get("http://www.domain.com");
// 转码前内容
echo requests::$content;
// 转码后内容
echo requests::$text;
```

网页状态码

可以用 requests::$status_code 来检查网页的状态码

```
requests::get("http://www.domain.com");
// 状态码
echo requests::$status_code;
// 如果是302跳转，获取跳转前状态码
echo requests::$history;
```

响应头内容

可以通过 requests::$headers 来获取响应头内容

```
requests::get("http://www.domain.com");
print_r(requests::$headers);
echo requests::$headers['Content-Type'];
```

请求头内容

可以通过 requests::$request['headers'] 来获取请求头内容。

```
requests::get("http://www.domain.com");
print_r(requests::$request['headers']);
```

自定义请求头部

伪装请求头部是采集时经常用的，我们可以用这个方法来隐藏：

```
requests::get("http://www.domain.com")
print_r(requests::$request['headers']['User-Agent']);
// php-requests/1.2.3 PHP/5.6 Windows/XP

requests::$request['headers']['User-Agent'] = 'phpspider';
requests::get("http://www.domain.com");
print_r(requests::$request['headers']['User-Agent']);
// phpspider
```

curl

curl -X PUT http://www.domain.com/demo.php -d "id=1" -d "title=a"

configs详解——之selector

`selector` 是页面元素选择器类，下面介绍此类可以调用的方法

`select($html, $selector, $selector_type = 'xpath')`

> @param $html 需筛选的网页内容
> @param $selector 选择器规则
> @param $selector_type 选择器类型: xpath、regex、css, 默认为xpath选择类型

栗子1:

通过xpath选择器提取网页内容的标题

```
$html = requests::get("http://www.epooll.com/archives/806/");
$data = selector::select($html, "//div[contains(@class,'page-header')]//h1//a");
var_dump($data);
```

栗子2:

通过css选择器提取网页内容的标题

```
$html = requests::get("http://www.epooll.com/archives/806/");
$data = selector::select($html, ".page-header > h1 > a", "css");
var_dump($data);
```

栗子3:

通过正则匹配提取网页内容的标题

```
$html = requests::get("http://www.epooll.com/archives/806/");
$data = selector::select($html, '@<title>(.*?)</title>@', "regex"
);
var_dump($data);
```

remove($html, $selector, $selector_type = 'xpath')

> @param $html 需过滤的网页内容
> @param $selector 选择器规则
> @param $selector_type 选择器类型: xpath、regex、css, 默认为xpath选择类型

举个例子:

```
$html =<<<STR
    <div id="demo">
        aaa
        <span class="tt">bbb</span>
        <span>ccc</span>
        <p>ddd</p>
    </div>
STR;

// 获取id为demo的div内容
$html = selector::select($html, "//div[contains(@id,'demo')]");
// 在上面获取内容基础上，删除class为tt的span标签
$data = selector::remove($html, "//span[contains(@class,'tt')]")
;
print_r($data);
```

configs详解——之db

本节介绍**db**类用法

数据库配置链接

```
$db_config = array(
    'host'  => '127.0.0.1',
    'port'  => 3306,
    'user'  => 'root',
    'pass'  => 'root',
    'name'  => 'qiushibaike',
);
// 数据库配置
db::set_connect('default', $db_config);
// 数据库链接
db::init_mysql();
```

原生SQL操作

query($sql)

举个栗子:

```
// 查询
$rsid = db::query("Select * From `content`");
while ( $row = db::fetch($rsid) )
{
    echo "id = {$row['id']}; name = {$row['name']}\n";
}

// 新增
db::query("Insert Into `content`(`name`) Value('test'));

// 更新
db::query("Update `content` Set `name`='test' Where `id`=1");

// 删除
db::query("Delete From `content` Where `id`='1'");
```

CRUD操作

get_one($sql)

单条查询

举个栗子:

```
$row = db::get_one("Select * From `content` Where `id`='1'");
```

get_all($sql)

多条查询

举个栗子:

```
$rows = db::get_all("Select * From `content` Limit 5");
```

insert($table, $data)

单条插入

举个栗子:

```
$data = array(
    'name' => 'test',
    'url'  => 'http://www.baidu.com'
);
$rows = db::insert('content', $data);
```

insert_batch($table, $data)

单条修改

举个栗子:

```
$data = array(
    array(
        'name' => 'test111',
        'url'  => 'http://www.baidu.com'
    ),
    array(
        'name' => 'test222',
        'url'  => 'http://www.baidu.com'
    ),
);
$rows = db::insert_batch('content', $data);
```

update_batch($table, $data, $index)

批量修改

举个栗子:

```
$data = array(
    array(
        'name' => 'test111',
        'url'  => 'http://www.baidu.com'
    ),
    array(
        'name' => 'test222',
        'url'  => 'http://www.baidu.com'
    ),
);
// 以name为条件进行修改
$rows = db::update_batch('content', $data, "name");
```

delete($table, $where)

单条删除

举个栗子:

```
$rows = db::delete('content', "`id`='1'");
```

configs详解——之log

`log` 对象提供不同级别的日志打印

`info($msg)`

打印普通日志

举个栗子:

```
log::info("成功处理一个页面");
```

`debug($msg)`

打印出错级别日志,调试时使用

举个栗子:

```
log::debug("正在提取文章标题数据");
```

`warn($msg)`

打印警告情况的日志

举个栗子:

```
log::warn("XX文件解析失败");
```

`error($msg)`

打印错误情况的日志

举个栗子:

```
log::error("XPath错误");
```

爬虫进阶开发——之内置方法

本节介绍爬虫的内置方法

add_url($url, $options = array())

一般在 `on_scan_page` 和 `on_list_page` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 用来往待爬队列中添加url

> @param $url 待添加的url
> @param $options 成员包括method, headers, params, context_data, reserve 和 proxy, 如下所示:
>
> > @param $options['method'] 默认为"get"请求, 也支持"post"请求
> > @param $options['headers'] 此url的Headers, 可以为空
> > @param $options['params'] 发送请求时需添加的参数, 可以为空
> > @param $options['context_data'] 此url附加的数据, 比如内容页需要列表页一些数据, 可以为空
> > @param $options['proxy'] 访问此url时使用的代理服务器，不使用请留空

栗子1:

```
$spider->on_scan_page = function($page, $content, $phpspider)
{
    $regex = "#http://pic.qiushibaike.com/system/pictures/\d+#";
    $urls = array();
    preg_match_all($regex, $content, $out);
    $urls = empty($out[0]) ? array() : $out[0];
    if (!empty($urls)) {
        foreach ($urls as $url)
        {
            $phpspider->add_url($url);
        }
    }
    ...
    return false;
};
```

add_scan_url($url, $options = array())

一般在 `on_start` 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 用来往待爬队列中添加scan_url

> @param $url 待添加的scan_url
> @param $options 成员包括method, headers, params, context_data, reserve 和 proxy, 如下所示:
>
> > @param $options['method'] 默认为"get"请求, 也支持"post"请求
> > @param $options['headers'] 此url的Headers, 可以为空
> > @param $options['params'] 发送请求时需添加的参数, 可以为空
> > @param $options['context_data'] 此url附加的数据, 比如内容页需要列表页一些数据, 可以为空
> > @param $options['proxy'] 访问此url时使用的代理服务器，不使用请留空

举个栗子:

```
$spider->on_start = function($page, $content, $phpspider)
{
    for ($i = 0; $i < 10; $i++)
    {
        $phpspider->add_scan_url("http://www.qiushibaike.com/page.php?page=".$i);
    }
};
```

栗子2:

```
$spider->on_list_page = function($page, $content, $phpspider)
{
    ...
    $next_url = str_replace($page['url'], "page=".$current_page_
num, "page=".$page_num);
    $options = array(
        'method' => 'post',
        'params' => array(
            'page' => $page_num,
            'size' => 18
        )
    );
    $phpspider->add_url($next_url, $options);
    return false;
};
```

request_url($url, $options = array())

一般在 on_start , on_download_page , on_scan_page 和 on_list_page 回调函数（在爬虫进阶开发——之回调函数中会详细描述）中调用, 下载网页, 得到网页内容

> @param $url 待添加的url
> @param $options 成员包括method, headers, params, context_data, reserve 和 proxy, 如下所示:
>
> > @param $options['method'] 默认为"get"请求, 也支持"post"请求
> > @param $options['headers'] 此url的Headers, 可以为空
> > @param $options['params'] 发送请求时需添加的参数, 可以为空
> > @param $options['context_data'] 此url附加的数据, 比如内容页需要列表页一些数据, 可以为空
> > @param $options['proxy'] 访问此url时使用的代理服务器，不使用请留空

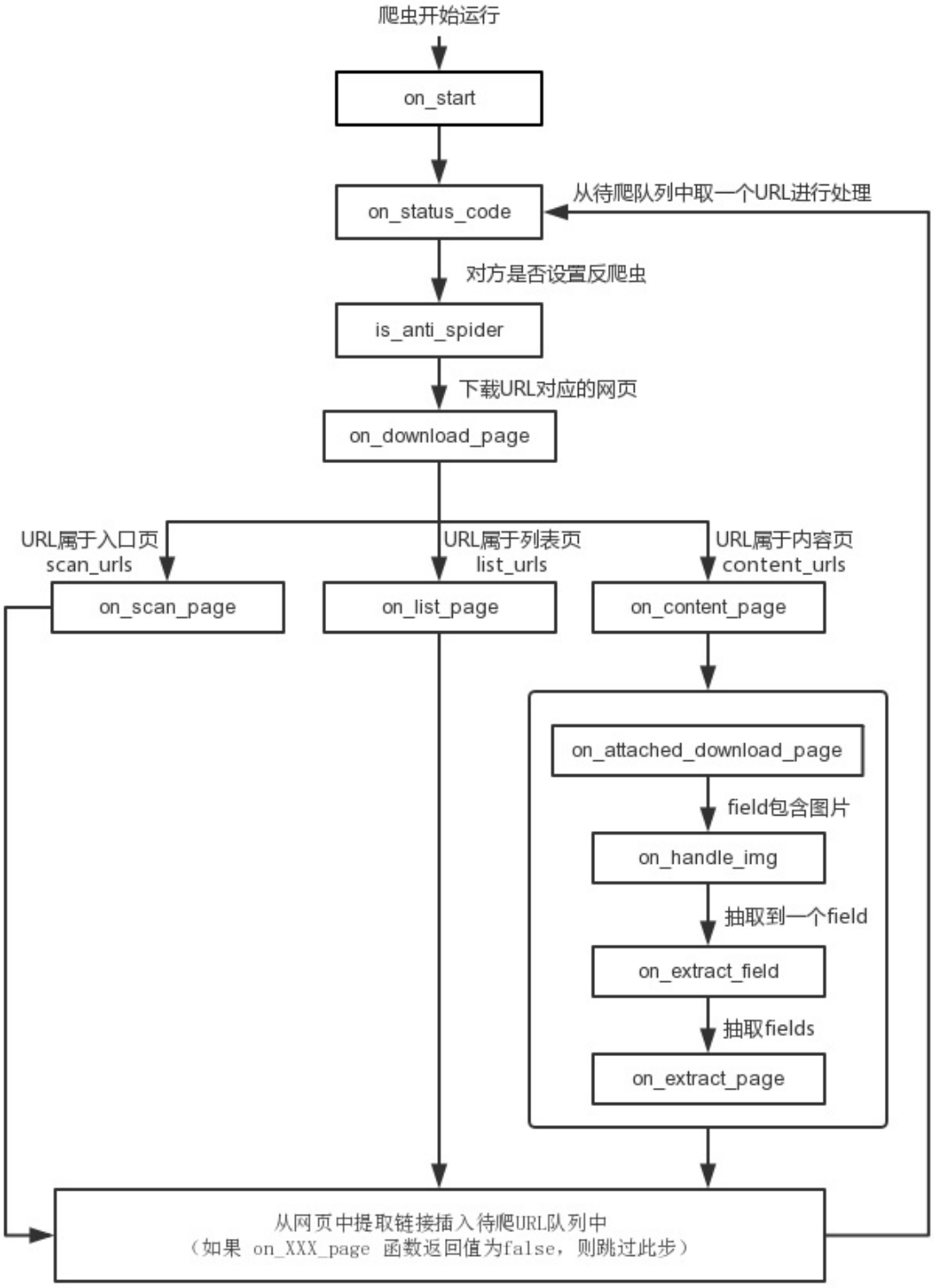
举个栗子:

```
$spider->on_download_page = function($page, $phpspider)
{
    $url = "https://checkcoverage.apple.com/cn/zh/?sn=FK1QPNCEGR
YD";
    $options = array(
        'method' => 'post',
        'params' => array(
            'sno' => 'FK1QPNCEGRYD',
            'CSRFToken' => $result[1]
        )
    );
    // 通过发送带参数的post请求，下载网页，并将网页内容赋值给result
    $request = $phpspider->request_url($url, $options);
    ...
    return $page;
};
```

爬虫进阶开发——之回调函数

回调函数是在爬虫爬取并处理网页的过程中设置的一些系统钩子,通过这些钩子可以完成一些特殊的处理逻辑.

下图是PHP蜘蛛爬虫爬取并处理网页的流程图，矩形方框中标识了爬虫运行过程中所使用的重要回调函数：



on_start($phpspider)

爬虫初始化时调用,用来指定一些爬取前的操作

> @param $phpspider 爬虫对象

在函数中可以调用 `requests::set_header($key,$value)`，`requests::set_cookie($key,$value)`，`requests::set_cookies($cookies)` 等

举个栗子:

在爬虫开始爬取之前给所有待爬网页添加一个Header

```
$spider->on_start = function($phpspider)
{
    requests::set_header("Referer", "http://buluo.qq.com/p/index
.html");
};
```

`on_status_code($status_code, $url, $content, $phpspider)`

判断当前网页是否被反爬虫了,需要开发者实现

> @param $status_code 当前网页的请求返回的HTTP状态码
> @param $url 当前网页URL
> @param $content 当前网页内容
> @param $phpspider 爬虫对象
> @return $content 返回处理后的网页内容，不处理当前页面请返回false

举个栗子:

```
$spider->on_status_code = function($status_code, $url, $content,
 $phpspider)
{
    // 如果状态码为429，说明对方网站设置了不让同一个客户端同时请求太多次
    if ($status_code == '429')
    {
        // 将url插入待爬的队列中,等待再次爬取
        $phpspider->add_url($url);
        // 当前页先不处理了
        return false;
    }
    // 不拦截的状态码这里记得要返回，否则后面内容就都空了
    return $content;
};
```

is_anti_spider($url, $content, $phpspider)

判断当前网页是否被反爬虫了,需要开发者实现

> @param $url 当前网页的url
> @param $content 当前网页内容
> @param $phpspider 爬虫对象
> @return 如果被反爬虫了, 返回true, 否则返回false

举个栗子:

```
$spider->is_anti_spider = function($url, $content, $phpspider)
{
    // $content中包含"404页面不存在"字符串
    if (strpos($content, "404页面不存在") !== false)
    {
        // 如果使用了代理IP，IP切换需要时间，这里可以添加到队列等下次换了IP再抓取
        // $phpspider->add_url($url);
        return true; // 告诉框架网页被反爬虫了，不要继续处理它
    }
    // 当前页面没有被反爬虫，可以继续处理
    return false;
};
```

on_download_page($page, $phpspider)

在一个网页下载完成之后调用.主要用来对下载的网页进行处理.

> @param $page 当前下载的网页页面的对象
> @param $phpspider 爬虫对象
> @return 返回处理后的网页内容
>
> > @param $page['url'] 当前网页的URL
> > @param $page['raw'] 当前网页的内容
> > @param $page['request'] 当前网页的请求对象

举个栗子:
比如下载了某个网页，希望向网页的body中添加html标签，处理过程如下：

```
$spider->on_download_page = function($page, $phpspider)
{
    $page_html = "<div id=\"comment-pages\"><span>5</span></div>
";
    $index = strpos($page['row'], "</body>");
    $page['raw'] = substr($page['raw'], 0, $index) . $page_html
. substr($page['raw'], $index);
    return $page;
};
```

on_download_attached_page($content, $phpspider)

在一个网页下载完成之后调用.主要用来对下载的网页进行处理.

> @param $content 当前下载的网页内容
> @param $phpspider 爬虫对象
> @return 返回处理后的网页内容

举个栗子:
比如下载的网页需去掉前后两边的中括号，并将处理后的数据返回，处理过程如下：

```
$spider->on_download_attached_page = function($content, $phpspid
er)
{
    $content = trim($content);
    $content = ltrim($content, "[");
    $content = rtrim($content, "]");
    $content = json_decode($content, true);
    return $content;
};
```

on_fetch_url($url, $phpspider)

在一个网页获取到**URL**之后调用.主要用来对获取到的**URL**进行处理.

> @param $url 当前获取到的URL
> @param $phpspider 爬虫对象
> @return 返回处理后的URL，为false则此URL不入采集队列

栗子1:
如果获取到URL包含#filter，不入URL采集队列

```
$spider->on_fetch_url = function($url, $phpspider)
{
    if (strpos($url, "#filter") !== false)
    {
        return false;
    }
    return $url;
};
```

栗子2:
把获取到URL的&替换成&，并将处理后的数据返回

```
$spider->on_fetch_url = function($url, $phpspider)
{
    $url = str_replace("&amp;", "&", $url);
    return $url;
};
```

on_scan_page($page, $content, $phpspider)

在爬取到入口**url**的内容之后, 添加新的**url**到待爬队列之前调用. 主要用来发现新的待爬**url**, 并且能给新发现的**url**附加数据（点此查看**“url**附加数据**”**实例解析）.

> @param $page 当前下载的网页页面的对象
> @param $content 当前网页内容
> @param $phpspider 当前爬虫对象
> @return 返回false表示不需要再从此网页中发现待爬url
>
> > @param $page['url'] 当前网页的URL
> > @param $page['raw'] 当前网页的内容
> > @param $page['request'] 当前网页的请求对象

此函数中通过调用$phpspider->add_url($url, $options)函数来添加新的url到待爬队列。

栗子1:
实现这个回调函数并返回false，表示爬虫在处理这个scan_url的时候，不会从中提取待爬url

```
$spider->on_scan_page = function($page, $content, $phpspider)
{
    return false;
};
```

栗子2:
生成一个新的url添加到待爬队列中，并通知爬虫不再从当前网页中发现待爬url

```
$spider->on_scan_page = function($page, $content, $phpspider)
{
    $array = json_decode($page['raw'], true);
    foreach ($array as $v)
    {
        $lastid = $v['id'];
        // 生成一个新的url
        $url = $page['url'] . $lastid;
        // 将新的url插入待爬的队列中
        $phpspider->add_url($url);
    }
    // 通知爬虫不再从当前网页中发现待爬url
    return false;
};
```

on_list_page($page, $content, $phpspider)

在爬取到入口**url**的内容之后, 添加新的**url**到待爬队列之前调用. 主要用来发现新的待爬**url**, 并且能给新发现的**url**附加数据（点此查看**“url**附加数据**”**实例解析）.

> @param $page 当前下载的网页页面的对象
> @param $content 当前网页内容
> @param $phpspider 当前爬虫对象
> @return 返回false表示不需要再从此网页中发现待爬url
>
> > @param $page['url'] 当前网页的URL
> > @param $page['raw'] 当前网页的内容
> > @param $page['request'] 当前网页的请求对象

此函数中通过调用$phpspider->add_url($url, $options)函数来添加新的url到待爬队列。

栗子1:
实现这个回调函数并返回false，表示爬虫在处理这个scan_url的时候，不会从中提取待爬url

```
$spider->on_list_page = function($page, $content, $phpspider)
{
    return false;
};
```

栗子2:
生成一个新的url添加到待爬队列中，并通知爬虫不再从当前网页中发现待爬url

```
$spider->on_list_page = function($page, $content, $phpspider)
{
    $array = json_decode($page['raw'], true);
    foreach ($array as $v)
    {
        $lastid = $v['id'];
        // 生成一个新的url
        $url = $page['url'] . $lastid;
        // 将新的url插入待爬的队列中
        $phpspider->add_url($url);
    }
    // 通知爬虫不再从当前网页中发现待爬url
    return false;
};
```

on_content_page($page, $content, $phpspider)

在爬取到入口**url**的内容之后, 添加新的**url**到待爬队列之前调用. 主要用来发现新的待爬**url**, 并且能给新发现的**url**附加数据（点此查看**“url**附加数据**”**实例解析）.

> @param $page 当前下载的网页页面的对象
> @param $content 当前网页内容
> @param $phpspider 当前爬虫对象
> @return 返回false表示不需要再从此网页中发现待爬url
>
> > @param $page['url'] 当前网页的URL
> > @param $page['raw'] 当前网页的内容
> > @param $page['request'] 当前网页的请求对象

此函数中通过调用$phpspider->add_url($url, $options)函数来添加新的url到待爬队列。

栗子1:
实现这个回调函数并返回false，表示爬虫在处理这个scan_url的时候，不会从中提取待爬url

```
$spider->on_content_page = function($page, $content, $phpspider)

{
    return false;
};
```

栗子2:
生成一个新的url添加到待爬队列中，并通知爬虫不再从当前网页中发现待爬url

```
$spider->on_content_page = function($page, $content, $phpspider)

{
    $array = json_decode($page['raw'], true);
    foreach ($array as $v)
    {
        $lastid = $v['id'];
        // 生成一个新的url
        $url = $page['url'] . $lastid;
        // 将新的url插入待爬的队列中
        $phpspider->add_url($url);
    }
    // 通知爬虫不再从当前网页中发现待爬url
    return false;
};
```

on_handle_img($fieldname, $img)

在抽取到**field**内容之后调用, 对其中包含的**img**标签进行回调处理

> @param $fieldname 当前field的name. 注意: 子field的name会带着父field的name, 通过.连接.
> @param $img 整个img标签的内容
> @return 返回处理后的img标签的内容

很多网站对图片作了延迟加载, 这时候就需要在这个函数里面来处理

举个栗子:
汽车之家论坛帖子的图片大部分是延迟加载的，默认会使用 `http://x.autoimg.cn/club/lazyload.png` 图片url，我们需要找到真实的图片url并替换，具体实现如下：

```
$spider->on_handle_img = function($fieldname, $img)
{
    $regex = '/src="(https?:\/\/.*?)"/i';
    preg_match($regex, $img, $rs);
    if (!$rs)
    {
        return $img;
    }
    $url = $rs[1];
    if ($url == "http://x.autoimg.cn/club/lazyload.png")
    {
        $regex2 = '/src9="(https?:\/\/.*?)"/i';
        preg_match($regex, $img, $rs);
        // 替换成真是图片url
        if (!$rs)
        {
            $new_url = $rs[1];
            $img = str_replace($url, $new_url);
        }
    }
    return $img;
};
```

on_extract_field($fieldname, $data, $page)

当一个field的内容被抽取到后进行的回调, 在此回调中可以对网页中抽取的内容作进一步处理

> @param $fieldname 当前field的name. 注意: 子field的name会带着父field的name, 通过.连接.
> @param $data 当前field抽取到的数据. 如果该field是repeated, data为数组类型, 否则是String
> @param $page 当前下载的网页页面的对象
> @return 返回处理后的数据, 注意数据类型需要跟传进来的 `$data` 类型匹配
>
> > @param $page['url'] 当前网页的URL
> > @param $page['raw'] 当前网页的内容
> > @param $page['request'] 当前网页的请求对象

举个栗子:
比如爬取知乎用户的性别信息, 相关网页源码如下:

```
 <span class="item gender" ><i class="icon icon-profile-male"></i>
</span>
```

那么可以这样写:

```
$configs = array(
    // configs的其他成员
    'fields' => array(
        array(
            'name' => "gender",
            'selector' => "//span[contains(@class, 'gender')]/i/
@class",
        ),
        ...
    )
);

$spider->on_extract_field = function($fieldname, $data, $page)
{
    if ($fieldname == 'gender')
    {
        // data中包含"icon-profile-male"，说明当前知乎用户是男性
        if (strpos($data, "icon-profile-male") !== false)
        {
            return "男";
        }
        // data中包含"icon-profile-female"，说明当前知乎用户是女性
        elseif (strpos($data, "icon-profile-female") !== false)
        {
            return "女";
        }
        else
        {
            return "未知";
        }
    }
    return $data;
};
```

on_extract_page($page, $data)

在一个网页的所有field抽取完成之后, 可能需要对field进一步处理, 以发布到自己的网站

> @param $page 当前下载的网页页面的对象
> @param $data 当前网页抽取出来的所有field的数据
> @return 返回处理后的数据, 注意数据类型需要跟传进来的 `$data` 类型匹配
>
> > @param $page['url'] 当前网页的URL
> > @param $page['raw'] 当前网页的内容
> > @param $page['request'] 当前网页的请求对象

举个栗子:
比如从某网页中得到time和title两个field抽取项, 现在希望把time的值添加中括号后拼凑到title中，处理过程如下：

```
$spider->on_extract_page = function($page, $data)
{
    $title = "[{$data['time']}]" . $data['title'];
    $data['title'] = $title;
    return $data;
    // 返回false不处理，当前页面的字段不入数据库直接过滤
    // 比如采集电影网站，标题匹配到“预告片”这三个字就过滤
    //if (strpos($data['title'], "预告片") !== false)
    //{
    //    return false;
    //}
};
```

爬虫进阶开发——xpath选择器常见用法

俗话说，工欲上其事，必先利其器，学好xpath选择器，能极高的提升在爬虫的数据提取环节中的提取速度，下面我们来认识认识xpath。

选取节点

XPath 使用路径表达式在 XML 文档中选取节点。节点是通过沿着路径或者 step 来选取的。

下面列出了最有用的路径表达式

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。

实例

1、精确查询

```
$html =<<<STR
    <div id="demo">
        <span class="tt">bbb</span>
        <span>ccc</span>
        <p rel="pnode">ddd</p>
    </div>
STR;

// 获取id为demo的div内容
$data = selector::select($html, "//div[@id='demo']");
// 获取class为tt的span内容
$data = selector::select($html, "//div[@class='tt']");
// 获取rel为pnode的p内容
$data = selector::select($html, "//div[@rel='pnode']");
```

2、模糊查询

contains 匹配一个属性值中包含的字符串

```
$html =<<<STR
    <div id="demo1">
        demo1
    </div>
    <div id="demo2">
        demo2
    </div>
STR;

// 查找id属性中包含demo关键字的页面元素
// 这里能获取id为demo1和demo2的内容
$data = selector::select($html, "//div[contains(@id,'demo')]");
```

3、获取节点属性

```
$html =<<<STR
    <td data-value="3.80">3.80</td>
    <td data-value="3.80">3.80</td>
    <td data-value="3.80">3.80</td>
    <td data-value="3.80">3.80</td>
STR;

// 获取 td 的 data-value 属性
$data = selector::select($html, "//td@data-value");
```

XPATH的几个常用函数

1.contains ()： //div[contains(@id, 'in')] ,表示选择id中包含有’in’的div节点

2.text()：由于一个节点的文本值不属于属性，比如 `<a class=”baidu“ href=”http://www.baidu.com“>baidu</a>` ,所以，用text()函数来匹配节点：//a[text()='baidu']

3.last()：//div[contains(@id, 'in')][las()]，表示选择id中包含有'in'的div节点的最后一个节点

4.starts-with()： //div[starts-with(@id, 'in')] ，表示选择以’in’开头的id属性的div节点

5.not()函数，表示否定，//input[@name=‘identity’ and not(contains(@class,‘a’))] ，表示匹配出name为identity并且class的值中不包含a的input节点。 not()函数通常与返回值为true or false的函数组合起来用，比如contains(),starts-with()等，但有一种特别情况请注意一下：我们要匹配出input节点含有id属性的，写法如下：//input[@id]，如果我们要匹配出input节点不含用id属性的，则为：//input[not(@id)]

爬虫进阶开发——之技巧篇

本节是开发爬虫模板时需要了解的技巧。包括，在爬取网站过程中经常遇到的问题，回调函数和内置函数的使用技巧等。

如何进行运行前测试？

> 在运行爬虫框架前，我们可能需要做很多准备工作
> 比如：登录验证测试、内容提取规则测试
> 这个时候我们就可以把PHPSpider当做类库来使用，获取单页面HTML并测试提取规则

内容提取测试

接下来我们以epooll这个站点的谋篇文章为例来演示内容提取方法

获取**HTML**内容

```
$url = "http://www.epooll.com/archives/806/";
$html = requests::get($url);
```

提取文章标题

```
// 选择器规则
$selector = "//div[contains(@class,'page-header')]//h1/a";
// 提取结果
$result = selector::select($html, $selector);
echo $result;
```

提取文章作者

```
$selector = "//div[contains(@class,'page-header')]//h6/span[1]";
$result = selector::select($html, $selector);
// 处理数据
$result = str_replace("作者：", "", $result);
echo $result;
```

提取文章入库完整示例

```
$url = "http://www.epooll.com/archives/806/";
$html = requests::get($url);

// 抽取文章标题
$selector = "//div[contains(@class,'page-header')]//h1/a";
$title = selector::select($html, $selector);
// 检查是否抽取到标题
//echo $title;exit;

// 抽取文章作者
$selector = "//div[contains(@class,'page-header')]//h6/span[1]";
$author = selector::select($html, $selector);
// 检查是否抽取到作者
//echo $author;exit;
// 去掉 作者：
$author = str_replace("作者：", "", $author);

// 抽取文章内容
$selector = "//div[contains(@class,'entry-content')]";
$content = selector::select($html, $selector);
// 检查是否抽取到内容
//echo $author;exit;

$data = array(
    'title' => $title,
    'author' => $author,
    'content' => $content,
);

// 查看数据是否正常
//print_r($data);

// 入库
db::insert("content", $data);
```

运行**PHPSpider**

通过上面的测试，我们就找出了文章内容页的 `field` 规则，配置到 `fields` ，然后调用PHPSpider

```
'fields' => array(
    // 文章标题
    array(
        'name' => "article_title",
        'selector' => "//div[contains(@class,'page-header')]//h1
/a",
        'required' => true,
    ),
    // 文章作者
    array(
        'name' => "article_author",
        'selector' => "//div[contains(@class,'page-header')]//h6
/span[1]",
        'required' => true,
    ),
    // 文章内容
    array(
        'name' => "article_content",
        'selector' => "//div[contains(@class,'entry-content')]",
        'required' => true,
    ),
)
```

如何实现模拟登录？

> 通过模拟登录, 可以解决登录后才能爬取某些网站数据的问题.
> PHPSpider框架提供两种登录方式：
>
> > 1、通过发送HTTP请求来实现模拟登录
> > 2、从Chrome浏览器拷贝Cookie字符串

通过发送**HTTP**请求来实现模拟登录

举个栗子:

```
// 登录请求url
$login_url = "http://www.waduanzi.com/login?url=http%3A%2F%2Fwww
.waduanzi.com%2F";
// 提交的参数
$params = array(
    "LoginForm[returnUrl]" => "http%3A%2F%2Fwww.waduanzi.com%2F"
,
    "LoginForm[username]" => "13712899314",
    "LoginForm[password]" => "854230",
    "yt0" => "登录",
);
// 发送登录请求
requests::post($login_url, $params);
// 登录成功后本框架会把Cookie保存到www.waduanzi.com域名下，我们可以看看
是否是已经收集到Cookie了
$cookies = requests::get_cookies("www.waduanzi.com");
print_r($cookies);  // 可以看到已经输出Cookie数组结构

// requests对象自动收集Cookie，访问这个域名下的URL会自动带上
// 接下来我们来访问一个需要登录后才能看到的页面
$url = "http://www.waduanzi.com/member";
$html = requests::get($url);
echo $html;     // 可以看到登录后的页面，非常棒
```

如何获得提交参数？

登录需要登录验证信息，下面我们来看看如何获得一个网站所需要的登录信息
还是以挖段子(www.waduanzi.com)为例，看看如何获得下面的信息

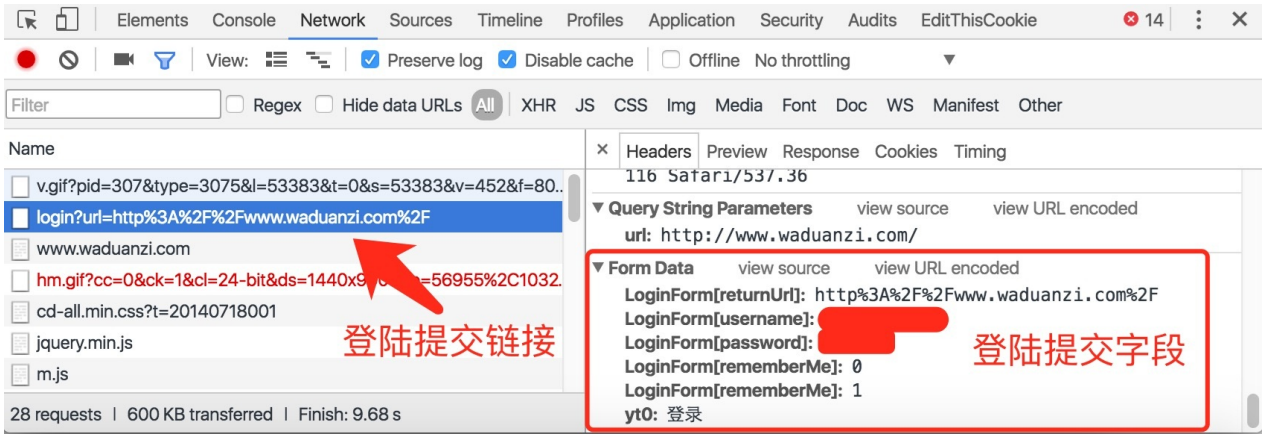
1、打开挖段子网站点击登录按钮进入登陆页：

```
http://www.waduanzi.com/login?url=http%3A%2F%2Fwww.waduanzi.com%2F
```

2、鼠标点击右键 **->** 检查 从而打开**Chrome**浏览器的开发者工具

选择**Network**选项卡，勾选**Preserve log**选项

3、填写登陆信息点击登录按钮，得到登录验证**URL**



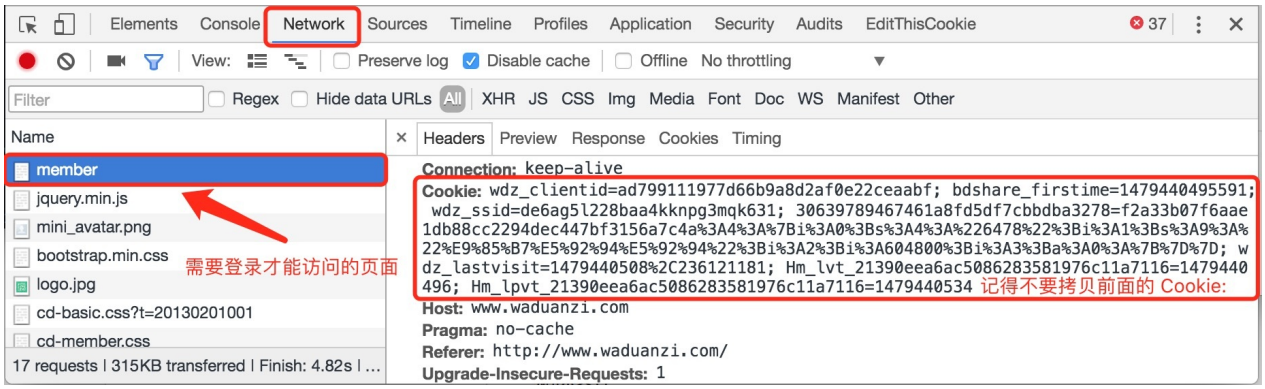
4、上面的登录提交字段填入框架代码

```
$params => array(
    "LoginForm[returnUrl]" => "http%3A%2F%2Fwww.waduanzi.com%2F"
,
    "LoginForm[username]" => "用户名",
    "LoginForm[password]" => "密码",
    "yt0" => "登录",
)
```

从Chrome浏览器拷贝Cookie字符串

上面的方式适用于简单的登录验证方式，如果遇到验证码，token表单字段，还有各种用js加密算法生成的登录字段，模拟登录就会变得异常复杂，为了节省时间，我们一般人工登录后，拷贝Cookie字符串来登录

1、登录成功后，在跳转的页面找**Cookie**字符串



2、上面的登录提交字段填入框架代码

```
// 模拟登录
$cookies = "复制上面Cookie:后面那一串字符串...";
requests::set_cookies($cookies, 'www.waduanzi.com');

// 接下来我们来访问一个需要登录后才能看到的页面
$url = "http://www.waduanzi.com/member";
$html = requests::get($url);
echo $html;      // 可以看到登录后的页面，非常棒
```

如何实现增量采集？

> 默认情况下，入口URL、列表URL和内容URL这所有的URL都有去重机制，就会对增量采集造成一定的麻烦。
> 框架开放了 add_scan_url() 接口，让用户可以在一次完整采集过后，添加新的入口URL(比如之前的入口URL、最新列表URL)来进行增量采集。
> 通过 add_scan_url() 方法添加的URL，不会被框架去重，从而达到增量采集的效果。。。

举个栗子:
我已经把糗事百科一次性采集完了，而糗百的内容更新都在首页，所以我可以在一次完整采集以后，把首页加入增量采集

```
$spider->on_start = function($phpspider)
{
    // add_sacn_url 没有URL去重机制，可用作增量更新
    $phpspider->add_scan_url("http://www.qiushibaike.com/");
};
```

如果内容页有分页，该如何爬取到完整数据？

> 如果要爬取的某个内容页中有多个分页，该如何爬取这个内容页的完整数据呢？这里就无法使用 `on_list_page` 回调函数了，而需要使用 `field` 中的 `attached_url` 来请求其他分页的数据。

举个栗子:
爬取某网站文章时，发现有些文章有多个内容页面，处理过程如下：

```
$configs = array(
    // configs 的其他成员
    ...
    'fields' => array(
        array(
            'name' => "contents",
            'selector' => "//div[@id='pages']//a//@href",
            'repeated' => true,
            'children' => array(
                array(
                    // 抽取出其他分页的url待用
                    'name' => 'content_page_url',
                    'selector' => "//text()"
                ),
                array(
                    // 抽取其他分页的内容
                    'name' => 'page_content',
                    // 发送 attached_url 请求获取其他的分页数据
                    // attached_url 使用了上面抓取的 content_page_u
rl
                    'source_type' => 'attached_url',
                    'attached_url' => 'content_page_url',
                    'selector' => "//*[@id='big-pic']"
                ),
            ),
        ),
    ),
);
```

在爬取到所有的分页数据之后，可以在 `on_extract_page` 回调函数中将这些数据组合成完整的数据

```
$spider->on_extract_field = function($fieldname, $data, $page)
{
    if ($fieldname == 'contents')
    {
        if (!empty($data))
        {
            $contents = $data;
            $data = "";
            foreach ($contents as $content)
            {
                $data .= $content['page_content'];
            }
        }
    }
    return $data;
};
```

如何实现多任务爬虫？

天下爬虫，唯快不破，配合多进程使用，phpspider可以快到你怕，下面我们来看看如何实现一个多任爬虫.

举个栗子：
同时开启8个任务

```
$configs = array(
    'name' => '糗事百科测试样例',
    'tasknum' => 8,    // 爬虫任务数
    ...
);
$spider = new phpspider($configs);
$spider->start();
```

运行界面：

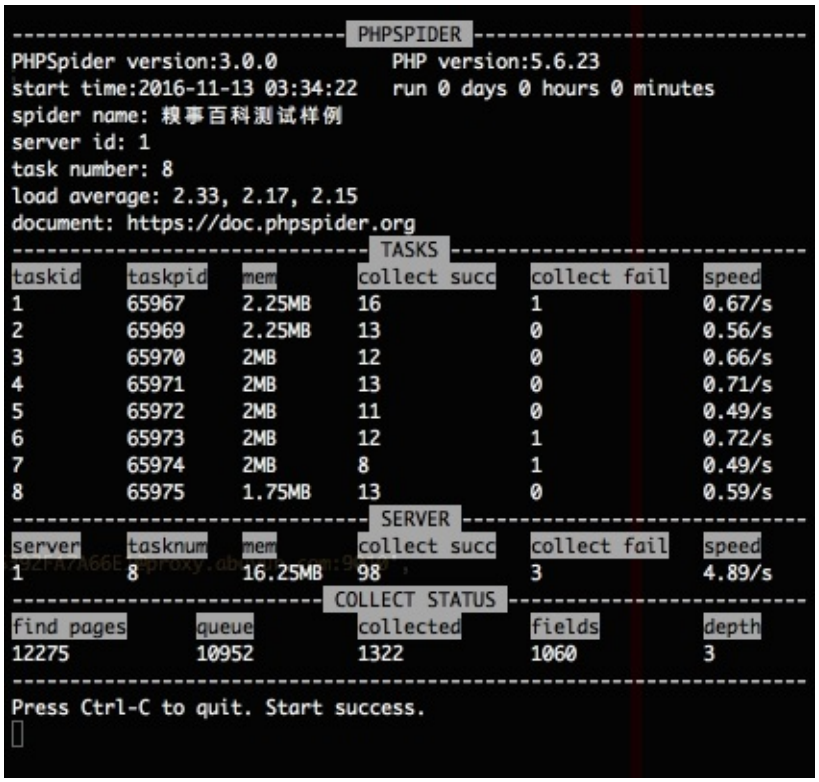
如何实现多服务器集群爬虫？

> 很多时候，单机器爬取的效率并不高，对于京东、淘宝这种动则上千万页面的网站，真的会爬到天荒地老，如何快速爬取成了当今爬虫最难的课题，要说破解防盗页面以及内容正则匹配提取，真的是特别的小儿科。
> 现在PHPSpider框架自带了集群功能，可以让初学者很轻易的在多台机器上运行同一分代码实现多机器爬取。

下面我们看看运行多任务爬虫所需要的代码

```
$configs = array(
    'name' => '糗事百科测试样例',
    'multiserver' => true,  // 是否启动集群爬虫
    'serverid' => 1,        // 集群服务器ID
    ...
);
$spider = new phpspider($configs);
$spider->start();
```

运行界面：



file_get_contents 设置代理抓取页面

普通页面获取

```
$url = "http://www.epooll.com/archives/806/";
$contents = file_get_contents($url);
preg_match_all("/<h1>(.*?)</h1>/is", $content, $matchs);
print_r($matchs[0]);
```

设置代理IP去采集数据

```
$context = array(
    'http' => array(
        'proxy' => 'tcp://192.168.0.2:3128',  //这里设置你要使用的代理ip及端口号
        'request_fulluri' => true,
    ),
);
$context = stream_context_create($context);
$html = file_get_contents("http://www.epooll.com/archives/806/",
 false, $context);
echo $html;
```

设置需要验证的代理IP去采集数据

```
$auth = base64_encode('USER:PASS');   //LOGIN:PASSWORD 这里是代理服务器的账户名及密码
$context = array(
    'http' => array(
        'proxy' => 'tcp://192.168.0.2:3128',  //这里设置你要使用的代理ip及端口号
        'request_fulluri' => true,
        'header' => "Proxy-Authorization: Basic $auth",
    ),
);
$context = stream_context_create($context);
$html = file_get_contents("http://www.epooll.com/archives/806/",
 false, $context);
echo $html;
```

如何提前生成列表页URL再提取内容？

> 通常情况下，爬虫会从起始页(scan_urls)开始通过列表页规则(list_url_regexes)寻找列表页，内容页同理，但是很多时候，第三方网站为了防止采集，会采用ajax的方式，不把列表页直接显式放在页面内容，而是通过js生成，又或者是直接显示前10页，因为正常的用户也只需要浏览前10页的数据就够了，现在我们针对这两种方式来看看抓取方法

只显示前10页的网页我们可以先生成列表页URL入队列

```
$configs = array(
    // configs的其他成员
    ...
    'scan_urls' => array(
        'https://www.itjuzi.com/investfirm?user_id=305129'
    ),
    'list_url_regexes' => array(
        "https://www.itjuzi.com/investfirm\?user_id=305129&page=
\d+"
    ),
    ...
);

$spider->on_start = function ($spider)
{
    // 生成列表页URL入队列
    for ($i = 0; $i <= 652; $i++)
    {
        $url = "https://www.itjuzi.com/investfirm?user_id=305129
&page={$i}";
        $spider->add_url($url);
    }
};
```

如何去掉网页中的广告?

> 当成功爬取到的网页数据中有很多不相干的html广告标签时, 你是否会感到无可奈何, 有时候即使将XPath的效果发挥到极致, 也无法去掉顽固的html广告标签, 咋整呢?

本节给你介绍通过selector类的remove方法去除html广告标签, 可提取有用数据或清理无用数据.

举个栗子:

在爬取某论坛问答帖时, 发现有很多html广告标签以及一些无用数据, 就需要在on_extract_field回调函数中调用selector的remove方法了

```
$configs = array(
    // configs的其他成员
    ...
    'fields' => array(
        array(
            'name' => "question_detail",
            'selector' => "XXX",
        ),
    ),
);
$spider->on_extract_field = function($fieldname, $data, $page)
{
    if ($fieldname == 'question_detail')
    {
        // 将data中符合XPath: "//div[contains(@class,'a_pr')]"的数据去掉
        $data = selector::remove($data, "//div[contains(@class,'a_pr')]");
        return $data;
    }
};
```

有时, 如果无用数据太多, 最好调用selector的select方法直接将有用的数据提取出来, 这么做会比调用remove方法更加方便.

如何爬取列表页中的数据?

> 一般情况下, 我们只需爬取内容页的数据即可, 不过有时候列表网页中也会有需要爬取的数据, 那想要爬取这部分数据, $phpspider->add_url($url, $options)函数

举个栗子:

在爬取爱游网的时候, 除了基本的内容页信息外, 还需要爬取浏览次数(或阅读量), 但是这些数据在列表页中, 这就需要在on_list_page回调函数中做处理

```
$configs = array(
    // configs的其他成员
    ...
    'fields' => array(
        array(
            'name' => "question_view_count",
            // 在内容页中通过XPath提取浏览次数(或阅读量)
            'selector' => "//a[contains(@class,'page-view')]",
            'required' => true,
        ),
    ),
);

$spider->on_list_page = function($page, $content, $phpspider)
{
    // 在列表页中通过XPath提取到内容页URL
    $content_url = selector::select($content, "//a[contains(@cla
ss,'s xst')]/@href");
    // 在列表页中通过XPath提取到浏览次数(或阅读量)
    $page_views = selector::select($content, "//td[contains(@cla
ss,'num')]/em");
    // 拼出包含浏览次数(或阅读量)的HTML代码
    $page_views = '<div><a class="page-view">' . $page_views + '
</a></div>';

    $options = array(
        'method' => 'get',
        'context_data' => $page_views,
    );

    $phpspider->add_url($content_url, $options);
    // 返回true继续提取其他列表页URL
    return true;
};
```

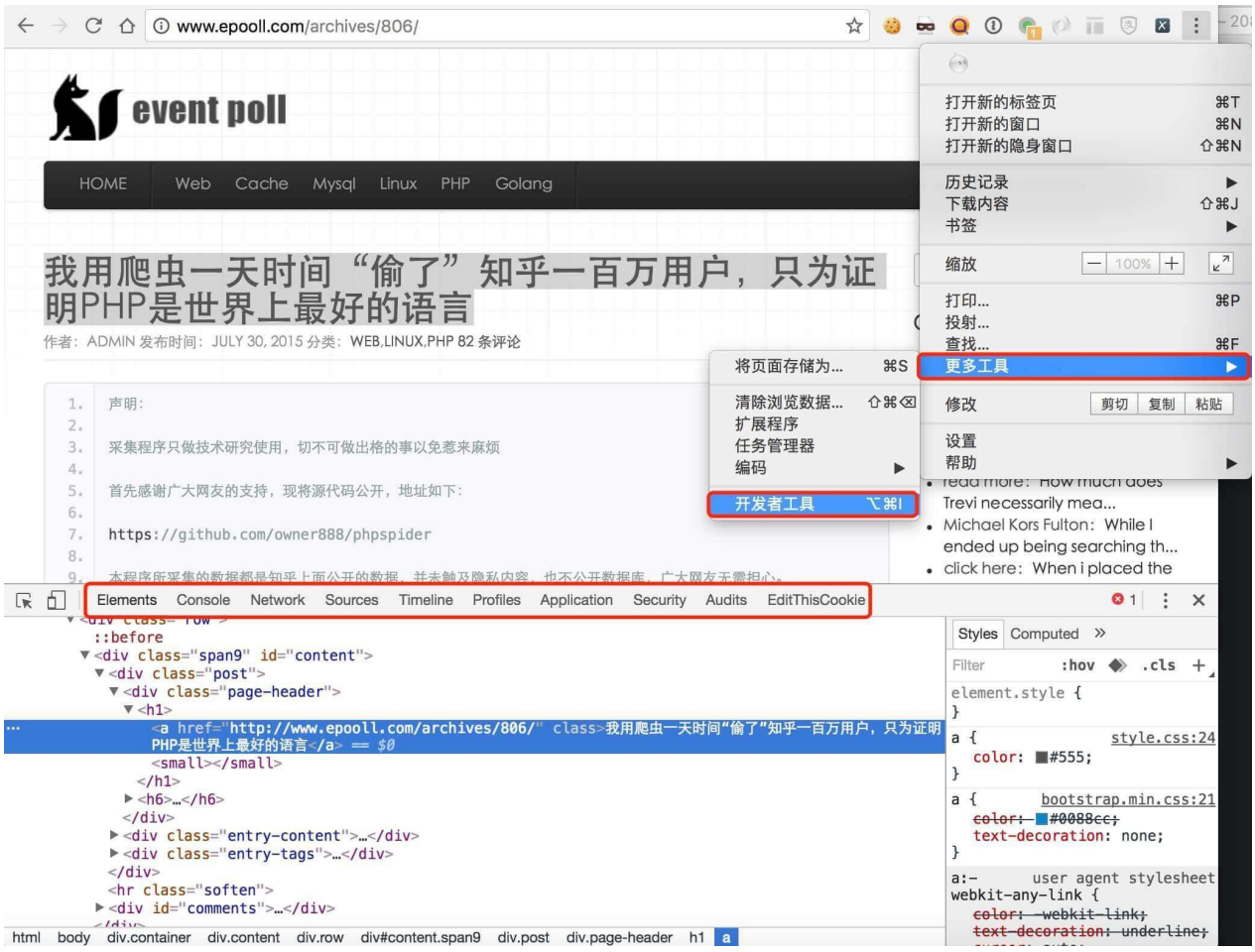
开发PHPSpider爬虫的常用工具

> “工欲善其事，必先利其器”，开发PHPSpider爬虫，起码得有几件顺手的工具才行吧，接下来给你逐个介绍。

谷歌Chrome浏览器

说起谷歌的Chrome浏览器（以下简称Chrome），相信大家都耳熟能详了吧，不仅使用流畅，而且功能强大，对开发PHPSpider爬虫非常有帮助。

我们主要使用的是Chrome的开发者工具，如下图所示：



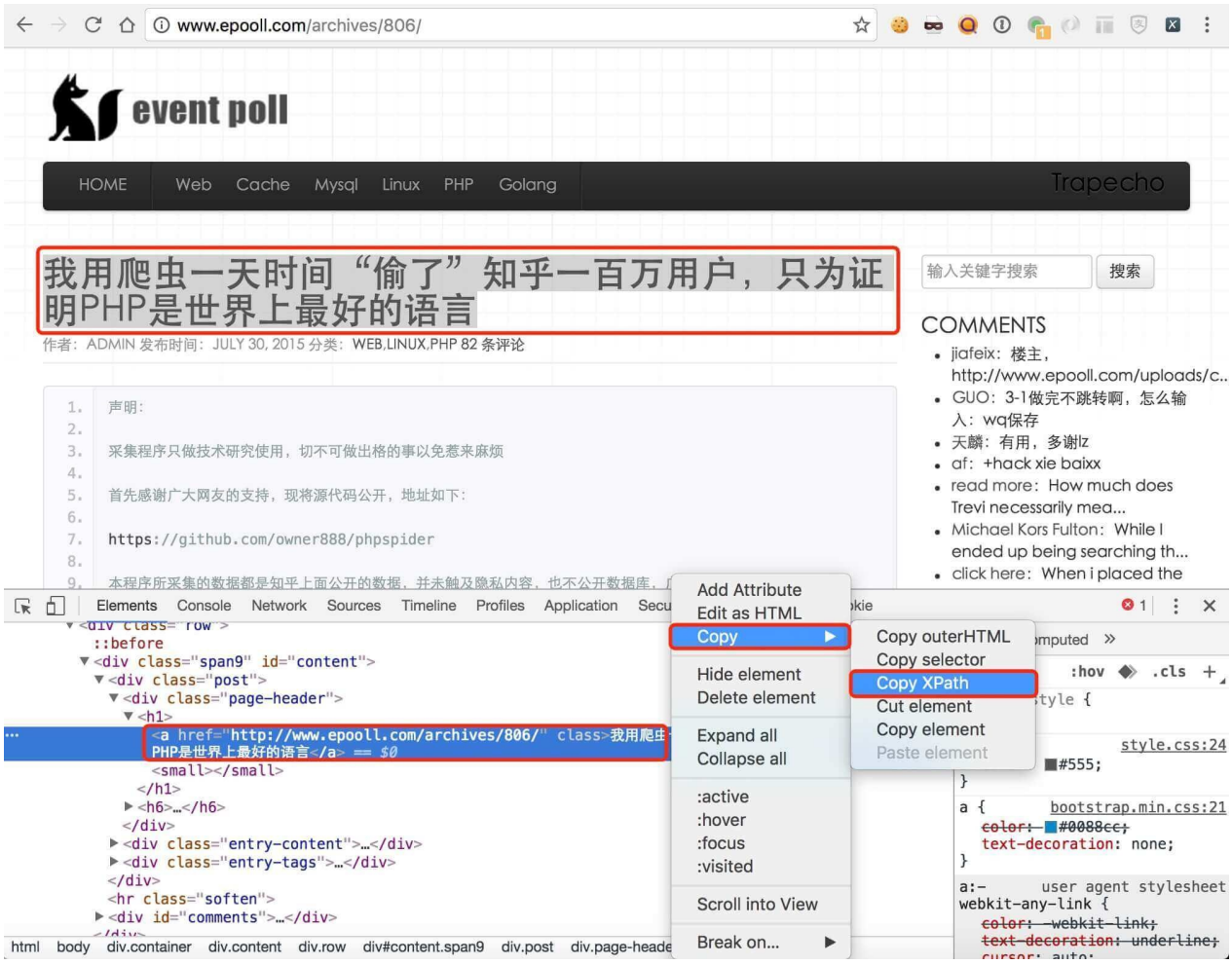
或者可以直接在网页上点击鼠标右键，选择“检查”，也可打开开发者工具。

开发者工具顶部有Elements、Console、Network等八个栏目。常用的有三个：Elements，用来查看需爬取字段的HTML标签信息；Console，可以检测你的JS代码；Network，用来分析HTTP请求。

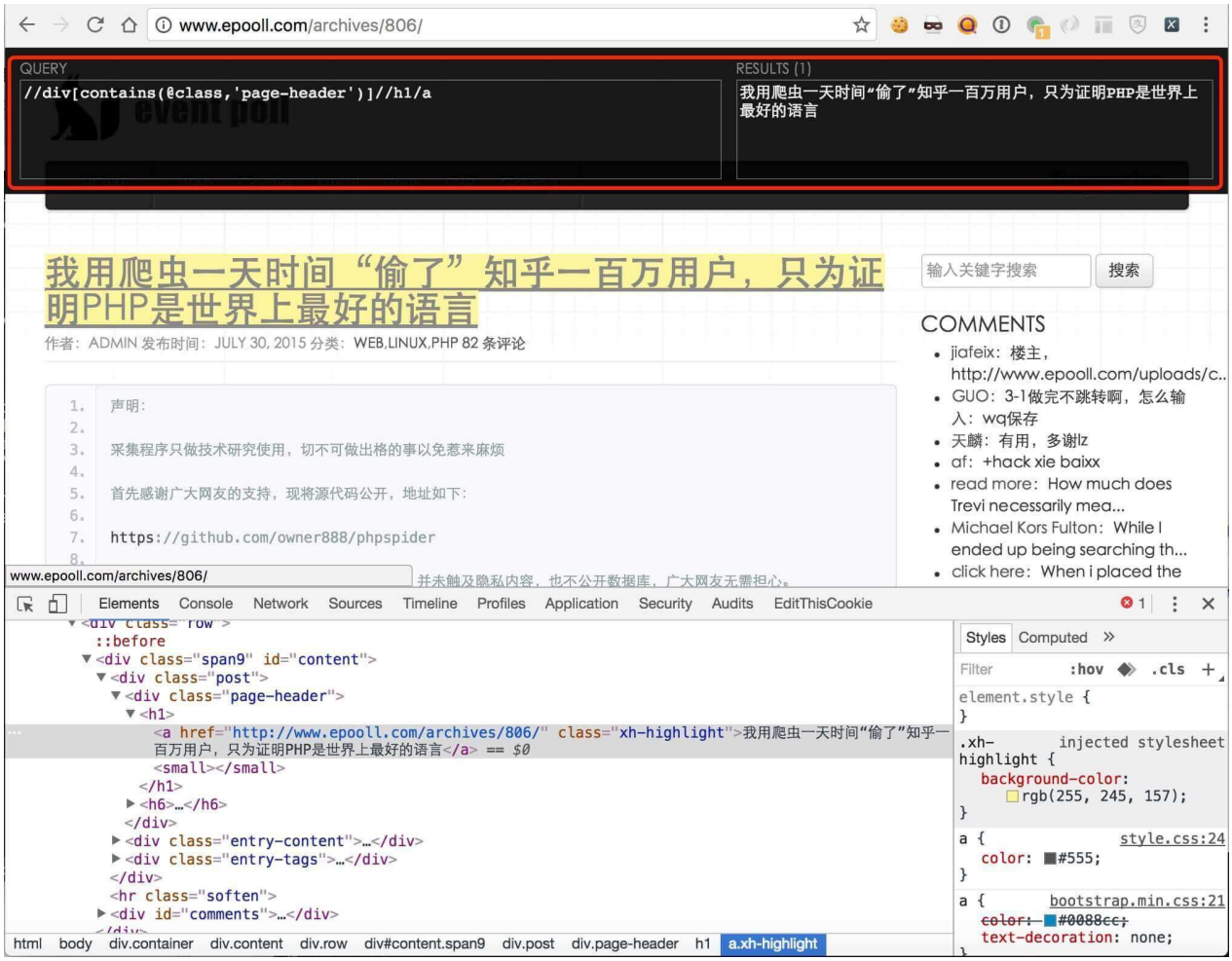
XPath Helper

XPath Helper是Chrome浏览器的插件，可以在Chrome应用商店安装下载，主要用来分析当前网页信息的XPath，并将其精简化。具体操作步骤如下：

1、在Chrome浏览器上，选择抽取的html字段并右击，点击“检查”，即可弹出开发者工具；右击已选字段，点击Copy XPath即可将该字段的XPath保存到浏览器剪贴板上，如下图所示：



2、打开XPath Helper插件，将得到的XPath复制进去，最好进行简化修改后再使用，如下图所示：



3、在XPath中，如果使用class属性来定位元素，最好使用contains函数，因为元素可能含有多个class：

```
(
    "name" => "article_title",
    "selector" => "//div[contains(@class,'page-header')]//h1/a"
),
```

4、在XPath中，如果使用id属性来定位元素，因为理论上id是唯一的，可以直接使用*[@id='']：

```
(
    "name" => "article_content",
    "selector" => "//*[@id='single-next-link']"
),
```

DHC REST

DHC REST也是Chrome浏览器的插件，可以在Chrome应用商店安装下载，主要用来模拟HTTP客户端发送测试数据到服务器。HTTP Get请求在开发中比较常用。

正则表达式测试工具

推荐使用站长工具中的正则表达式测试工具，链接如下：
http://tool.chinaz.com/regex/