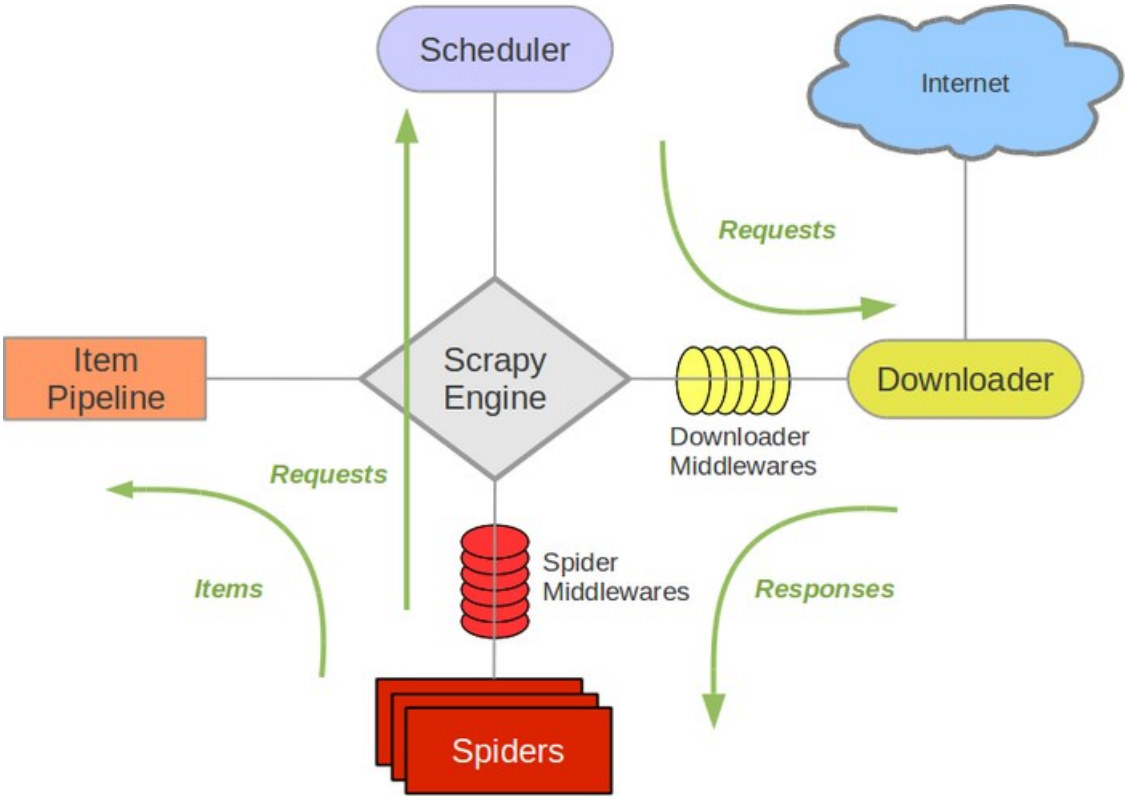
scrapy

Scrapy 框架官方网址：http://doc.scrapy.org/en/latest

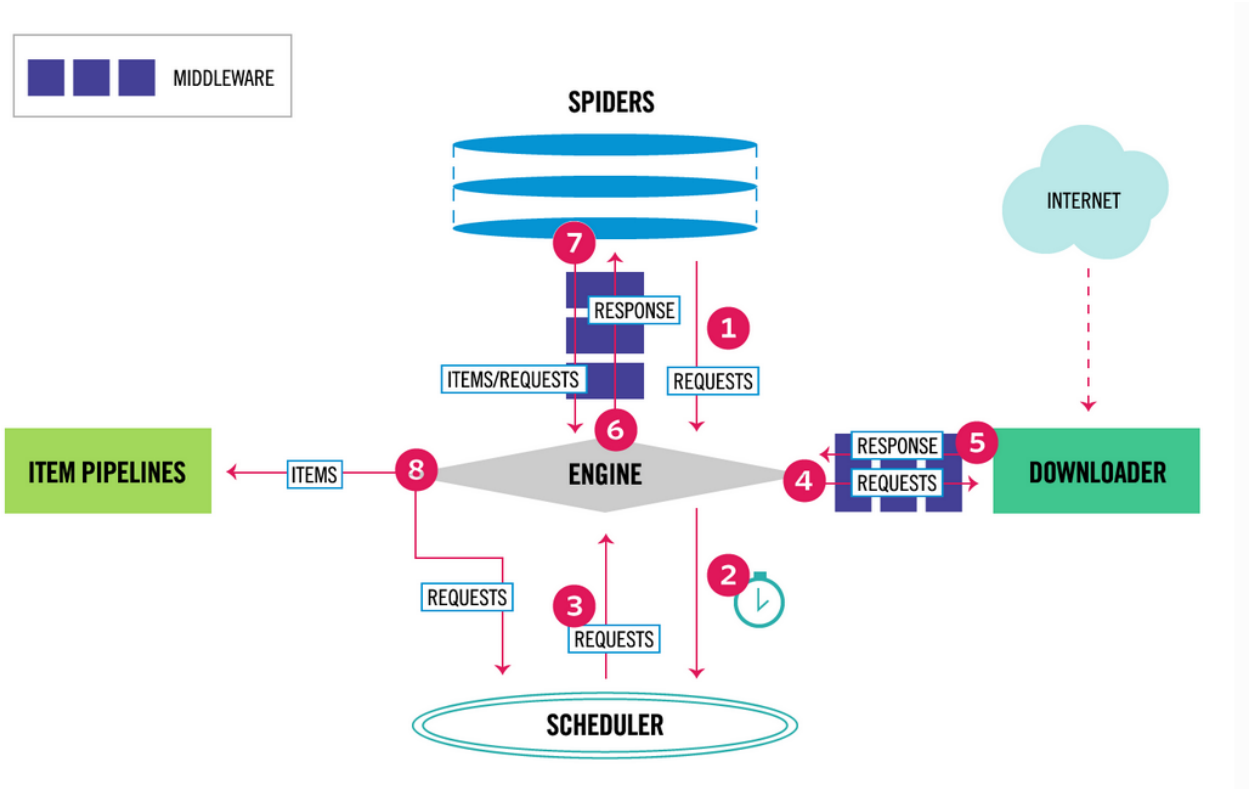
Scrapy 中文维护站点：http://scrapy-chs.readthedocs.io/zh_CN/latest/index.html

- Scrapy 是用纯 Python 实现一个为了爬取网站数据、提取结构性数据而编写的应用框架，用途非常广泛。
- 框架的力量，用户只需要定制开发几个模块就可以轻松的实现一个爬虫，用来抓取网页内容以及各种图片，非常之方便。
- Scrapy 使用了 Twisted['twistid](其主要对手是 Tornado)异步网络框架来处理网络通讯，可以加快我们的下载速度，不用自己去实现异步框架，并且包含了各种中间件接口，可以灵活的完成各种需求。



- Scrapy Engine(引擎): 负责 Spider、ItemPipeline、Downloader、Scheduler 中间的通讯，信号、数据传递等。
- Scheduler(调度器): 它负责接受引擎发送过来的 Request 请求，并按照一定的方式进行整理排列，入队，当引擎需要时，交还给引擎。
- Downloader（下载器）：负责下载 Scrapy Engine(引擎)发送的所有 Requests 请求，并将其获取到的 Responses 交还给 Scrapy Engine(引擎)，由引擎交给 Spider 来处理，

- Spider（爬虫）：它负责处理所有 Responses,从中分析提取数据，获取 Item 字段需要的数据，并将需要跟进的 URL 提交给引擎，再次进入 Scheduler(调度器)，
- Item Pipeline(管道)：它负责处理 Spider 中获取到的 Item，并进行进行后期处理（详细分析、过滤、存储等）的地方.
- Downloader Middlewares（下载中间件）：你可以当作是一个可以自定义扩展下载功能的组件。
- Spider Middlewares（Spider 中间件）：你可以理解为是一个可以自定扩展和操作引擎和 Spider 中间通信的功能组件（比如进入 Spider 的 Responses;和从 Spider 出去的 Requests）



scrapy 运行流程：

1. 爬虫引擎获得初始请求开始抓取。
2. 爬虫引擎开始请求调度程序，并准备对下一次的请求进行抓取。
3. 爬虫调度器返回下一个请求给爬虫引擎。
4. 引擎请求发送到下载器，通过下载中间件下载网络数据。
5. 一旦下载器完成页面下载，将下载结果返回给爬虫引擎。
6. 引擎将下载器的响应通过中间件返回给爬虫进行处理。
7. 爬虫处理响应，并通过中间件返回处理后的 items，以及新的请求给引擎。
8. 引擎发送处理后的 items 到项目管道，然后把处理结果返回给调度器，调度器计划处理下一个请求抓取。

9. 重复该过程（继续步骤 1），直到爬取完所有的 url 请求。

Scrapy 爬虫步骤：

1. 新建项目 (scrapy startproject xxx)：新建一个新的爬虫项目
2. 明确目标（编写 items.py）：明确你想要抓取的目标
3. 制作爬虫（spiders/xxspider.py）：制作爬虫开始爬取网页
4. 存储内容（pipelines.py）：设计管道存储爬取内容

scrapy Shell

Selectors 选择器

Scrapy Selectors 内置 XPath 和 CSS Selector 表达式机制

Selector 有四个基本的方法，最常用的还是 xpath:

- `xpath()`: 传入 xpath 表达式，返回该表达式所对应的所有节点的 selector list 列表
- `extract()`: 序列化该节点为 Unicode 字符串并返回 list
- `css()`: 传入 CSS 表达式，返回该表达式所对应的所有节点的 selector list 列表，语法同 BeautifulSoup4
- `re()`: 根据传入的正则表达式对数据进行提取，返回 Unicode 字符串 list 列表

XPath 表达式的例子及对应的含义:

`/html/head/title`: 选择`<HTML>`文档中 `<head>` 标签内的 `<title>` 元素
`/html/head/title/text()`: 选择上面提到的 `<title>` 元素的文字
`//td`: 选择所有的 `<td>` 元素
`//div[@class="mine"]`: 选择所有具有 `class="mine"` 属性的 `div` 元素

Selector 使用

```
from scrapy import Selector

body = '<html><head><title>Hello python</title></head></html>'

selector = Selector(text=body)
title = selector.xpath('//title/text()').extract_first()
print(title)
```

```
Hello python
```

我们在这里没有在 scrapy 框架中运行，而是把 scrapy 中的 selector 单独拿出来了，构建的时候传入 rext 参数，就生成一个 selector 选择器对象 以后做数据提取的时候，可以把现在 Scrapy Shell 中测试，测试通过后再应用到代码中。

当然 Scrapy Shell 作用不仅仅如此，但是不属于我们课程重点，不做详细介绍。

官方文档：http://scrapy-chs.readthedocs.io/zh_CN/latest/topics/shell.html

如果安装了 IPython ，Scrapy 终端将使用 IPython (替代标准 Python 终端)。 IPython 终端与其他相比更为强大，提供智能的自动补全，高亮输出，及其他特性。（推荐安装 IPython）

启动 Scrapy Shell

进入项目的根目录，执行下列命令来启动 shell:

```
scrapy shell "url"
```

Scrapy Shell 根据下载的页面会自动创建一些方便使用的对象，例如 Response 对象，以及 Selector 对象 (对 HTML 及 XML 内容)。

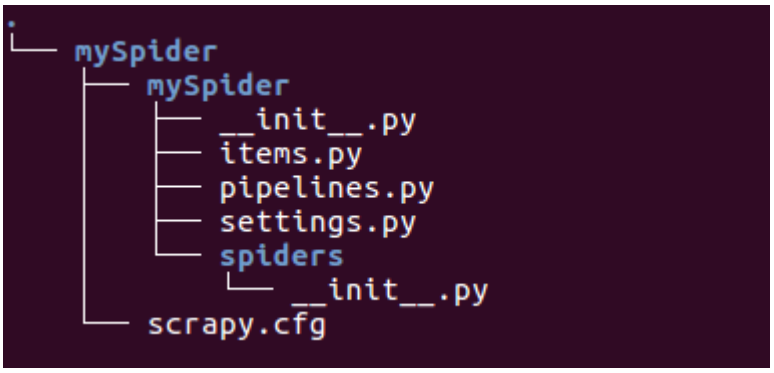
- 当 shell 载入后，将得到一个包含 response 数据的本地 response 变量，输入 `response.body` 将输出 response 的包体，输出 `response.headers` 可以看到 response 的包头。
- 输入 `response.selector` 时， 将获取到一个 response 初始化的类 Selector 的对象，此时可以通过使用 `response.selector.xpath()`或 `response.selector.css()` 来对 response 进行查询。
- Scrapy 也提供了一些快捷方式, 例如 `response.xpath()`或 `response.css()`同样可以生效（如之前的案例）。

scrapy 入门实例

新建项目(scrapy startproject)

- 在开始爬取之前，必须创建一个新的 Scrapy 项目。进入自定义的项目目录中，运行下列命令：

```
scrapy startproject 项目名
```



- scrapy.cfg ： 项目的配置文件
- mySpider/ ： 项目的 Python 模块，将会从这里引用代码
- mySpider/items.py ： 项目的目标文件
- mySpider/pipelines.py ： 项目的管道文件
- mySpider/settings.py ： 项目的设置文件
- mySpider/spiders/ ： 存储爬虫代码目录

创建爬虫 （spiders）

spider 是自己定义的类，scrapy 用它来从网页里抓取内容，并解析抓取的结果，这个类必须继承 scrapy.Spider，定义 spider 的名称和起始请求，包括了对页面的请求以及页面的处理.

在当前目录下输入命令，在目录下创建一个名为 quotes 的爬虫，并指定爬取域的范围：

```
scrapy genspider quotes quotes.toscrape.com
```

spider 目录里多了一个 quotes.py 文件，代码如下:

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = 'quotes'
    allowed_domains = ['quotes.toscrape.com']
    start_urls = ['http://quotes.toscrape.com/']

    def parse(self, response):
        pass
```

其实也可以由我们自行创建 quotes.py 并编写上面的代码，只不过使用命令可以免去编写固定代码的麻烦,要建立一个 Spider， 你必须用 scrapy.Spider 类创建一个子类，并确定了三个强制的属性 和 一个方法。

- `name = ""` ：这个爬虫的识别名称，必须是唯一的，在不同的爬虫必须定义不同的名字。
- `allow_domains = []` 是搜索的域名范围，也就是爬虫的约束区域，规定爬虫只爬取这个域名下的网页，不存在的 URL 会被忽略。
- `start_urls = ()` ：爬取的 URL 元祖/列表。爬虫从这里开始抓取数据，所以，第一次下载的数据将会从这些 urls 开始。其他子 URL 将会从这些起始 URL 中继承性生成。
- `parse(self, response)` ：他是 spider 的一个解析方法，每个初始 URL 完成下载后将被调用，调用的时候传入从每一个 URL 传回的 Response 对象来作为唯一参数，主要作用如下：
    1. 负责解析返回的网页数据(response.body)，提取结构化数据(生成 item)
    2. 生成需要下一页的 URL 请求。

创建 item(mySpider/items.py)

1. 打开 mySpider 目录下的 `items.py`
2. Item 定义结构化数据字段，用来保存爬取到的数据，有点像 Python 中的 dict，但是提供了一些额外的保护减少错误。
3. 可以通过创建一个 `scrapy.Item` 类，并且定义类型为 `scrapy.Field` 的类属性来定义一个 Item（可以理解成类似于 ORM 的映射关系）。

接下来，创建一个 `QuotesSpider` 类，和构建 item 模型（model）。

```
import scrapy

class TutorialItem(scrapy.Item):
    text = scrapy.Field()
    author = scrapy.Field()
    tags = scrapy.Field()
```

解析 Response

爬取整个网页完毕，接下来的就是的取过程了，首先观察页面源码,直接用 XPath 开始提取数据吧。 我们之前在 tutorial/items.py 里定义了一个 QuotesSpider 类。 这里引入进来

```
from mySpider.items import QuotesSpider
```

然后将我们得到的数据封装到一个 QuotesSpider 对象中：

```
from mySpider.items import QuotesSpider

def parse(self, response):
        quotes = response.xpath("//div[@class='quote']")
        for quote in quotes:
            item = TutorialItem()
```

```
            item['text'] = quote.xpath("//span[@class='text']/text()").extract()
            item['author'] = quote.xpath("//small[@class='author']/text()").extra
ct()
            item['tags'] = quote.xpath("//a[@class='tag']/text()").extract()
            yield item
```

我们暂时先不处理管道，后面会详细介绍。

保存数据

scrapy 保存信息的最简单的方法主要有四种，-o 输出指定格式的文件，，命令如下：

```
# json 格式，默认为Unicode 编码
scrapy crawl quotes -o quotes.json

# json lines 格式，默认为Unicode 编码
scrapy crawl quotes -o quotes.jsonl

# csv 逗号表达式，可用Excel 打开
scrapy crawl quotes -o quotes.csv

# xml 格式
scrapy crawl quotes -o quotes.xml
```

Item Pipeline

当 Item 在 Spider 中被收集之后，它将会被传递到 Item Pipeline，这些 Item Pipeline 组件按定义的顺序处理 Item。

每个 Item Pipeline 都是实现了简单方法的 Python 类，比如决定此 Item 是丢弃而存储。以下是 item pipeline 的一些典型应用：

- 清理 HTML 数据
- 验证爬取的数据(检查 item 包含某些字段，比如说 name 字段)
- 查重并丢弃重复内容
- 将爬取结果保存到文件或者数据库中

编写 item pipeline

编写 item pipeline 很简单，item pipiline 组件是一个独立的 Python 类，其中 process_item() 方法必须实现:

```
import something

class SomethingPipeline(object):
    def __init__(self):
        # 可选实现，做参数初始化等
        # doing something

    def process_item(self, item, spider):
        # item (Item 对象) – 被爬取的item
        # spider (Spider 对象) – 爬取该item 的spider
        # 这个方法必须实现，每个item pipeline 组件都需要调用该方法，
        # 这个方法必须返回一个 Item 对象，被丢弃的item 将不会被之后的pipeline 组件所处
理。
        return item

    def open_spider(self, spider):
        # spider (Spider 对象) – 被开启的spider
        # 可选实现，当spider 被开启时，这个方法被调用。

    def close_spider(self, spider):
        # spider (Spider 对象) – 被关闭的spider
        # 可选实现，当spider 被关闭时，这个方法被调用
```

item 写入 JSON 文件(完善之前的案例)

以下 pipeline 将所有(从所有'spider'中)爬取到的 item，存储到一个独立地 items.json 文件，每行包含一个序列化为'JSON'格式的'item'。

打开 pipelines.py 文件，写入下面代码:

```
import json

class TutorialPipeline(object):

    def __init__(self):
        self.file = open('quotes.json', 'wb')

    def process_item(self, item, spider):
        content = json.dumps(dict(item), ensure_ascii=False) + "\n"
        self.file.write(content)
        return item

    def close_spider(self, spider):
        self.file.close()
```

Item 存入 MongoDB

首先确保 Mongodb 已经正常安装并且正常运行

```
import pymongo

class MongoPipeline(object):
    def __init__(self, mongo_uri, mongo_db):
        self.mongo_uri = mongo_uri
        self.mongo_db = mongo_db

    @classmethod
    def from_crawler(cls, crawler):
        return cls(
            mongo_uri=crawler.settings.get('MONGO_URI'),
            mongo_db=crawler.settings.get('MONGO_DB')
        )

    def open_spider(self, spider):
        self.client = pymongo.MongoClient(self.mongo_uri)
        self.db = self.client[self.mongo_db]

    def process_item(self, item, spider):
        name = item.collection
        self.db[name].insert(dict(item))
        return item

    def close_spider(self, spider):
        self.client.close()
```

这里需要用到两个变量，MONGO*URI 和 MONGO*DB，即存储到 MongoDB 的链接地址和数据库名称。 可以在 settings.py 里面添加这两个变量

```
MONGO_URI = 'locahost'
MONGO_DB = 'itemmongodb'
```

Item 存入 MySQL

首先确保 mysql 已经正常安装并且正常运行

新建数据库好数据库和数据表

```
import pymysql

class MysqlPipeline():
    def __init__(self, host, database, user, password, port):
```

```
        self.host = host
        self.database = database
        self.user = user
        self.password = password
        self.port = port

    @classmethod
    def from_crawler(cls, crawler):
        return cls(
            host=crawler.settings.get('MYSQL_HOST'),
            database=crawler.settings.get('MYSQL_DATABASE'),
            user=crawler.settings.get('MYSQL_USER'),
            password=crawler.settings.get('MYSQL_PASSWORD'),
            port=crawler.settings.get('MYSQL_PORT'),
        )

    def open_spider(self, spider):
        self.db = pymysql.connect(self.host, self.user, self.password, self.datab
ase, charset='utf8',
                                  port=self.port)
        self.cursor = self.db.cursor()

    def close_spider(self, spider):
        self.db.close()

    def process_item(self, item, spider):
        print(item['title'])
        data = dict(item)
        keys = ', '.join(data.keys())
        values = ', '.join(['%s'] * len(data))
        sql = 'insert into %s (%s) values (%s)' % (item.table, keys, values)
        self.cursor.execute(sql, tuple(data.values()))
        self.db.commit()
        return item
```

这里需要用到几个 mysql 配置。 可以在 settings.py 里面添加这些变量

```
MYSQL_HOST = 'localhost' # 地址
MYSQL_DATABASE = 'database' # 数据库名称
MYSQL_USER = 'root' # 用户名
MYSQL_PASSWORD = 'password' # 密码
MYSQL_PORT = '3306' # 端口
```

启用 Item Pipeline 组件

为了启用 Item Pipeline 组件，必须将它的类添加到 settings.py 文件 ITEM_PIPELINES 配置，就像下面这个例子:

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
    "tutorial.pipelines.TutorialPipeline":300,
    # "tutorial.pipelines.MongoPipeline":300,
    # "tutorial.pipelines.MysqlPipeline":300
}
```

分配给每个类的整型值，确定了他们运行的顺序，item 按数字从低到高的顺序，通过 pipeline，通常将这些数字定义在 0-1000 范围内（0-1000 随意设置，数值越低，组件的优先级越高）

Settings

Scrapy 设置(settings)提供了定制 Scrapy 组件的方法。可以控制包括核心(core)，插件(extension)，pipeline 及 spider 组件。比如 设置 Json Pipeliine、LOG_LEVEL 等。

参考文档：
http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html#topics-settings-ref

内置设置参考手册

- BOT_NAME
  - –默认: 'scrapybot'
  - –当您使用 startproject 命令创建项目时其也被自动赋值。
- CONCURRENT_ITEMS
  - –默认: 100
  - –Item Processor(即 Item Pipeline) 同时处理(每个 response 的)item 的最大值。
- CONCURRENT_REQUESTS
  - –默认: 16
  - –Scrapy downloader 并发请求(concurrent requests)的最大值。
- DEFAULT_REQUEST_HEADERS
  - –默认: 如下

```
{
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.
8',
    'Accept-Language': 'en',
}
```

Scrapy HTTP Request 使用的默认 header。

- DEPTH_LIMIT

  –默认: 0

  –爬取网站最大允许的深度(depth)值。如果为 0，则没有限制。

- DOWNLOAD_DELAY

  –默认: 0

  –下载器在下载同一个网站下一个页面前需要等待的时间。该选项可以用来限制爬取速度， 减轻服务器压力。同时也支持小数:

```
DOWNLOAD_DELAY = 0.25 # 250 ms of delay
```

  –默认情况下，Scrapy 在两个请求间不等待一个固定的值， 而是使用 0.5 到 1.5 之间的一个随机值 * DOWNLOAD_DELAY 的结果作为等待间隔。

- DOWNLOAD_TIMEOUT

  –默认: 180

  –下载器超时时间(单位: 秒)。

- ITEM_PIPELINES

  –默认: {}

  –保存项目中启用的 pipeline 及其顺序的字典。该字典默认为空，值(value)任意，不过值(value)习惯设置在 0-1000 范围内，值越小优先级越高。

```
ITEM_PIPELINES = {
    'mySpider.pipelines.SomethingPipeline': 300,
    'mySpider.pipelines.ItcastJsonPipeline': 800,
}
```

- LOG_ENABLED

  –默认: True

  –是否启用 logging。

- LOG_ENCODING

–默认: 'utf-8'

–logging 使用的编码。

- LOG_LEVEL

  –默认: 'DEBUG'

  –log 的最低级别。可选的级别有: CRITICAL、 ERROR、WARNING、INFO、DEBUG 。

- USER_AGENT

  –默认: "Scrapy/VERSION (+http://scrapy.org)"

  –爬取的默认 User-Agent，除非被覆盖。

- PROXIES： 代理设置

  –示例：

```
PROXIES = [
    {'ip_port': '111.11.228.75:80', 'password': ''},
    {'ip_port': '120.198.243.22:80', 'password': ''},
    {'ip_port': '111.8.60.9:8123', 'password': ''},
    {'ip_port': '101.71.27.120:80', 'password': ''},
    {'ip_port': '122.96.59.104:80', 'password': ''},
    {'ip_port': '122.224.249.122:8088', 'password':''},
]
```

- COOKIES_ENABLED = False

  –禁用 Cookies