Scrapy

- 引言

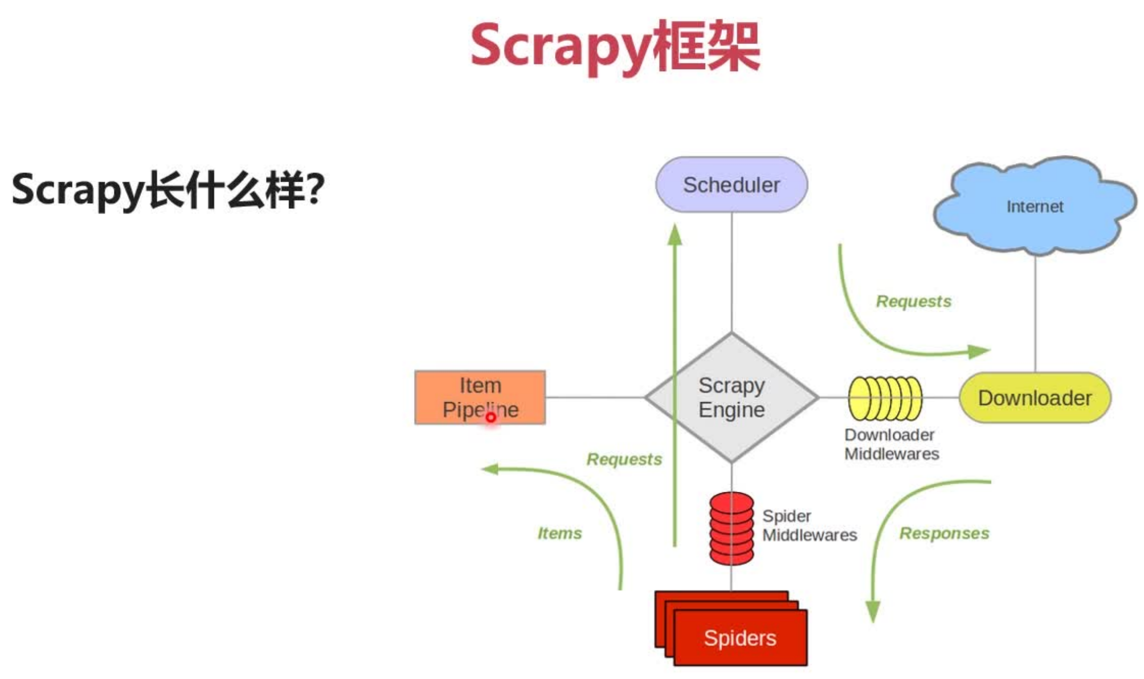
```
1. Scrapy 是爬虫框架之一
2. 完成了多线程取，并提供了很多内置的组件
3. 框架是代码的半成品（已经完成了大部分工作）
4. Scrapy的结构可以和其他的组件进行组合，实现特殊的操作
```

- Scrapy的安装

```
1. pip install scrapy

需要依赖：
1. vcpython
        C++库---（安装时报错）
2. win32api：
        windows用户---（运行时报错）---需要调用win指令
        不能直接用pip 安装
        wein32api是非官方的--- 需要先下载whl源码压缩包
        然后使用pip install  **.whl
3. Twisted
        运行时如果报错，缺少该模块
        pip install twisted
```

- 什么是Scrapy


Scrapy框架

Scrapy长什么样?

```
scrapy组件
1. Scrapy Engine
        Scrapy的引擎
        调度和调配所有其他组件
        scrapy的最核心代码
        封装死的，不可修改
2. Scheduler
        调度器
        从引擎接收request 并让requset请求入队，方便引擎调用
        引擎会将队列中的request 发送给下载器
3. Downloader:
        下载器，获取页面或其他响应，并生成response对象，返回给引擎，之后返回给Spider
4. Spider：
        解析器
        分析由下载器返回的response对象
        节点分析，json解析，js渲染的解析，二进制文件的处理
        将解析出来的数据，打包成item（必须是字典），item被引擎调用发送给PipeLines
        或将解析出来的新的url---被打包成request对象，返回给引擎，进一步返回给调度器
5. Item Pipelines
        项目管道
        接收Spider打包好的Item数据，逐个处理
        1. 数据清洗 2. 数据去重 3. 数据验证  4. 数据持久化
        可以有多个管道，多个管道用于执行不同的操作（对item）
        多个管道如何管理：
            设置权重值---规定了优先级---构建成了一种链式管道关系
6. Download Middlewares:
        下载器中间件
        解耦合
        下载器和引擎之间的耦合
        常用于：给请求加入头部信息，cookies信息，加入U-A认证
7. Spider Middlewares
        解析器中间件
        解耦合
        解析器和引擎之间的耦合
```

- 命令的使用

```
scrapy 本身较重
常用的命令：
1. 创建项目
        scrapy  startproject 项目名
2. 创建Spider爬虫
        scrapy genspider 爬虫名 入口地址
        入口地址：是一个url
3. 运行爬虫
        scrapy crawl 爬虫名
        启动爬虫--通过指令的形式
4. 验证shell脚本
        scrapy shell 网站url

        view(response) 跳转到浏览器查看网页
        response.text  查看网页源代码
```

- 案例：51招聘

```
1. 爬取的网站的类型
		招聘类---数据搜索型网站
2. 爬取的内容
		1. 列表页中详情页的url，考虑分页
		2. 详情页中具体的数据
3. 存储方式：
		mysql
4. 如何让爬虫持续运行
		1. 爬虫一直运行，不停止
		2. 做断点记录
		3. 定时任务

思路：
1. 搜索类的网站：
		以数据量为导向：
			地点：尽量选择全国
		以数据分析为导向：
			地点：分别进行爬取
2. 目标网站：
		51job
		https://search.51job.com
3. 分析：
		列表页：
			详情页的url：
		详情页：
			1. 公司名
			2. 职位名称
			3. 工资
			4. 基本要求
			5. 职位信息
			6. 联系方式
			7. 公司信息
			8. 公司规模
```

- 前置工作

```
1. scrapy  startproject job
		创建项目
2. scrapy genspider zhaopin search.51job.com
		创建爬虫
3. 技巧：
		在scrapy.cfg同级位置创建main.py文件
		编辑指令：
		import scrapy.cmdline
		scrapy.cmdline.execute(['scrapy','crawl','爬虫名'])
4. 注意：
		xpath会返回一个列表，列表的每个元素都是selector对象，如果要使用其内容，需要再次调用
extract（）方法
```

- spider

```
# -*- coding: utf-8 -*-
import scrapy
import com.baizhi.AI145.爬虫.scrapy的应用.job.job.items as items

# 入口
class ZhaopinSpider(scrapy.Spider):
    name = 'zhaopin'  # 爬虫名
    allowed_domains = ['search.51job.com','www.zhaopin.com']  # 允许访问的域名---检测
    start_urls = ['http://search.51job.com/'] # 入口链接---原则上应该是列表页的第一页
    headers={'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 
(KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36'}

    # 1. 引擎第一次调用star_urls 入口链接之后，该链接被封装成request对象，发送给了调度器
    # 2. 引擎从调度器拿到入口链接，给了下载器
    # 3. 下载器从网络端，发送了请求，拿到了响应，并打包成response对象发送给了引擎
    # 4. 引擎拿到response对象发送给Spider--ZhaopinSpider--parse接收---进行解析操作
    def parse(self, response):
        # 解析：scrapy支持xpath语法
        #     # 1. Selector对象  选择器对象
        #     resSelec=scrapy.Selector(response)
        #     # 2.
        #     print(resSelec.xpath('//*'))

        cities=['010000','090200']
        cindex=0
        kw=['python','AI']
        kindex=0
        while 1:
            # 1. 动态构建构建url
            url = 'https://search.51job.com/list/%s,000000,0000,00,9,99,%s,2,1.html'%
(cities[cindex], kw[kindex])

            # 2. 发送请求
            yield 
scrapy.Request(url=url,callback=self.listParse,headers=self.headers,dont_filter=True)
 # 回调函数，要解析新的响应
            # 循环断点


    # 解析列表页
    def listParse(self,response):
        # 初始化
        res=scrapy.Selector(response)
        # 详情页连接

        urls=res.xpath('//p[@class="t1 "]/span/a/@href').extract() # 序列化

        # 进一步访问详情页
        for i in urls:
            # i: 每个详情页的url
            yield 
scrapy.Request(url=i,callback=self.detailParse,headers=self.headers,dont_filter=True)
        nextPage=res.xpath('//li[@class="bk"]')[1].xpath('./a/@href').extract()
        print(nextPage[0],'*************')
        if nextPage:
            yield 
scrapy.Request(url=nextPage[0],callback=self.listParse,headers=self.headers,dont_filter
=True)

    def detailParse(self,response):
        # 解析详情页
        res=scrapy.Selector(response)
        item = items.JobItem()  # 是一个类字典对象
        item['jobName']=res.xpath('//h1/@title').extract()[0]
        item['salary']=res.xpath('//div[@class="cn"]/strong/text()').extract()[0]
        item['companyName']=res.xpath('//p[@class="cname"]/a/@title').extract()[0]
        item['baseReq']=res.xpath('//p[@class="msg ltype"]/@title').extract()[0]
        tempList=res.xpath('//div[@class="tCompany_main"]/div')
        item['jobReq']=tempList[0].xpath('./div//p/text()').extract()[0]
        item['addr']=tempList[1].xpath('./div/p/text()').extract()[0]
        item['info']=tempList[2].xpath('./div/text()').extract()[0]

        return  item
```

- items

```
# 实体类---entity
# 对应于库表（属性应该是库表的字段）
        #实体类的属性：尽量和库表中的字段名字一致
# item 对象作为数据承载的媒介

 # -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# https://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class JobItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    jobName=scrapy.Field()  # 描述符
    # 封装成一个字典
    salary=scrapy.Field()
    companyName=scrapy.Field()
    baseReq=scrapy.Field()
    jobReq=scrapy.Field()
    addr=scrapy.Field()
    info=scrapy.Field()
    a=scrapy.Field(HeheItem()) # 通过组合构成外键关系
```

```
class HeheItem(scrapy.Item): # 可以设置多个Item
    hehe=scrapy.Field()
```

- pipelines

```
1. 是管道，其中可以有多个pipeline对象,不同的管道对象有不同的名字和不同的权重值
        权重值：将多个管道构建成链式关系 ---排序
        权重值越大，优先级越低
        建议：权重值从300开始
2. scrapy的运行利用了：
        分布式+异步
        提高执行效率
3. 为了防止数据发生重复
        管道会构建链式结构---利用管道维护
4. 管道的数据丢弃
        1. 如果该管道操作已经执行过（不处理）
        2. 如果item数据无效（筛掉）
5. pipeline一定要记得在settings中设置
```

*

- 去重

```
class DeleteRepatPipeLine:
    def __init__(self):
        self.temp=set()
    def process_item(self, item, spider):
        s=item['jobName']+item['salary']+item['company']+item['baseReq']
        if s in self.temp:
            raise DropItem('该item已存在')
        else:
            self.temp.add(s)
            return item
```

- 清洗

```
# 清洗
class DataCleanPipline:
    def process_item(self,item,spider):
        for i in item:
            item[i].strip()
            # 去除所有不需要的代码
        item['info']=','.join(ja.extract_tags(item['info'],topK=10))
        item['jobReq']=','.join(ja.extract_tags(item['jobReq'],topK=10))
        return item
```

- 时间字符串转换

```
import re
import datetime
class DateFormatPipeline:
    def process_item(self,item,spider):
        # 转换成时间字符串
        item2={'publishTime':'4小时前'}
        item3={'publishTime':'1天前'}
        if '小时' in item2['publishTime']:
            item2['publishTime']=time.strftime('%Y-%m-%d', time.localtime())
        elif '天前' in item3['publishTime']:
            s=item3['publishTime']
            day=int(re.compile('\d*').findall(s)[0])
            item3['publishTime']=time.strftime('%Y-%m-%d', time.localtime(time.time()-
day*24*60*60))

        if '天前' in item3['publishTime']:
            s = item3['publishTime']
            day = int(re.compile('\d*').findall(s)[0])
            item3['publishTime'] = str(datetime.date.today()-
datetime.timedelta(days=day))
```

- 数据存储

```
1. 需要创建一个单独的表，该表的字段包含id和数据字段名
        数字:1字节/数  8位---  -2^7~2^7-1
        varchar: 2/字符  8位/字节  --16位
2. 只要是需要存储的字段，都必须经过判断
        目的为了防止重复
3. 库表：
        1. 主表：要存的字段的id值（尽量存id）
        2. n多的副表：只有两个字段： id -- 数据字段名
        目的：
            1. 提高主表的查询效率
            2. 极大的缩小主表的体量
            3. 有利于数据分析
4. 突出了表关系
```

```
# -*- coding: utf-8 -*-

# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: https://doc.scrapy.org/en/latest/topics/item-pipeline.html
import MySQLdb

class JobPipeline(object): # 相当于火车站    item相当于火车
    def __init__(self,host, port, user,password, db, charset):
        self.host=host
        self.port=port
        self.user=user
        self.password=password
        self.db=db
        self.charset=charset
    def process_item(self, item, spider):
        # 数据的处理
        # 入库
        cursor.execute('insert into 51job VALUES (%s,%s,%s,%s,%s,%s,%s)',
[item['jobName'],item['salary'],item['companyName'],item['baseReq'],

 item['jobReq'],item['addr'],item['info']])
        conn.commit()

    # spider开启时被调用--开启数据库
    def open_spider(self,spider):
        self.conn = MySQLdb.Connection(host=self.host, port=self.port, user=self.user,
password=self.password, db=self.db, charset=self.charset)
        self.cursor = conn.cursor()

    # spider关闭时被调用---关闭数据库
    def close_spider(self,spider):
        self.cursor.close()
        self.conn.close()


    # 创建pipeline对象之前调用
    # 引擎会自动调用该方法，且是第一个调用该方法---创建pipeline对象
    @classmethod
    def from_crawler(cls,crawler):
        return cls(host='localhost', port=3306, user='root',password='123456',
db='crawler', charset='utf8')
```

- settings

```
# -*- coding: utf-8 -*-

# Scrapy settings for job project
#
# For simplicity, this file contains only settings considered important or
# commonly used. You can find more settings consulting the documentation:
#
#     https://doc.scrapy.org/en/latest/topics/settings.html
#     https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
#     https://doc.scrapy.org/en/latest/topics/spider-middleware.html

BOT_NAME = 'job' # 项目名

SPIDER_MODULES = ['job.spiders'] # 爬虫模块
NEWSPIDER_MODULE = 'job.spiders' # 爬虫配置


# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'job (+http://www.yourdomain.com)'  # U-A  告知被爬取的服务器，我是爬虫我叫job
# 可以配置U-A

# Obey robots.txt rules
ROBOTSTXT_OBEY = False  # 是否要遵守君子协定

# Configure maximum concurrent requests performed by Scrapy (default: 16)
```

```python
#CONCURRENT_REQUESTS = 32

# Configure a delay for requests for the same website (default: 0)
# See https://doc.scrapy.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
#DOWNLOAD_DELAY = 3
# The download delay setting will honor only one of:
#CONCURRENT_REQUESTS_PER_DOMAIN = 16
#CONCURRENT_REQUESTS_PER_IP = 16

# Disable cookies (enabled by default)
COOKIES_ENABLED = False

# Disable Telnet Console (enabled by default)
#TELNETCONSOLE_ENABLED = False

# Override the default request headers:
#DEFAULT_REQUEST_HEADERS = {
#   'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
#   'Accept-Language': 'en',
#}

# Enable or disable spider middlewares
# See https://doc.scrapy.org/en/latest/topics/spider-middleware.html
#SPIDER_MIDDLEWARES = {
#    'job.middlewares.JobSpiderMiddleware': 543,
#}

# Enable or disable downloader middlewares
# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
#DOWNLOADER_MIDDLEWARES = {
#    'job.middlewares.JobDownloaderMiddleware': 543,
#}

# Enable or disable extensions
# See https://doc.scrapy.org/en/latest/topics/extensions.html
#EXTENSIONS = {
#    'scrapy.extensions.telnet.TelnetConsole': None,
#}

# Configure item pipelines
# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
   'job.pipelines.JobPipeline': 400,
   'job.pipelines.DeleteRepatPipeLine': 300,
   'job.pipelines.DataCleanPipline': 330,
   'job.pipelines.DateFormatPipeline': 340,
}

# Enable and configure the AutoThrottle extension (disabled by default)
# See https://doc.scrapy.org/en/latest/topics/autothrottle.html
#AUTOTHROTTLE_ENABLED = True
# The initial download delay
```

```
#AUTOTHROTTLE_START_DELAY = 5
# The maximum download delay to be set in case of high latencies
#AUTOTHROTTLE_MAX_DELAY = 60
# The average number of requests Scrapy should be sending in parallel to
# each remote server
#AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0
# Enable showing throttling stats for every response received:
#AUTOTHROTTLE_DEBUG = False

# Enable and configure HTTP caching (disabled by default)
# See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-middleware-settings
#HTTPCACHE_ENABLED = True
#HTTPCACHE_EXPIRATION_SECS = 0
#HTTPCACHE_DIR = 'httpcache'
#HTTPCACHE_IGNORE_HTTP_CODES = []
#HTTPCACHE_STORAGE = 'scrapy.extensions.httpcache.FilesystemCacheStorage'

MyHost='localhost'
MyPort=3306
MyUser='root'
MyPass='123456'
MyDB='crawler'
MyChar='utf8'
```

- 定时任务

```
1. windows
        控制面板-计划任务-新建任务
        1. 常规：名称，给最高权限，不需要登录即可运行
        2. 触发器：新建：频次，时间
        3. 操作：新建：操作--启动程序，脚本：选取  .bat脚本
                脚本内容：
                cd 项目目录
                scrapy crawl 爬虫名
        4. 条件：默认，不修改
        5. 设置：建议选择过期立即开启任务
2. linux
        crontab---定时任务
```

- middlewares

```
# -*- coding: utf-8 -*-

# Define here the models for your spider middleware
#
# See documentation in:
# https://doc.scrapy.org/en/latest/topics/spider-middleware.html

from scrapy import signals
```

```python
class JobSpiderMiddleware(object):
    # Not all methods need to be defined. If a method is not defined,
    # scrapy acts as if the spider middleware does not modify the
    # passed objects.

    # 创建Spider中间件对象之前调用（创建spider中间件）
    @classmethod
    def from_crawler(cls, crawler):
        # This method is used by Scrapy to create your spiders.
        s = cls()
        crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
        return s

    # 任何response进入到Spier之前调用该方法
    def process_spider_input(self, response, spider):
        # Called for each response that goes through the spider
        # middleware and into the spider.

        # Should return None or raise an exception.
        return None

    #  Spider返回任何结果时调用该方法
    def process_spider_output(self, response, result, spider):
        # Called with the results returned from the Spider, after
        # it has processed the response.

        # result 必须是一个可迭代对象，可迭代对象的元素必须是：Request, dict or Item
        # Must return an iterable of Request, dict or Item objects.
        # 返回的必须是一个可迭代对象，可迭代对象的元素必须是：Request, dict or Item
        for i in result:
            yield i

    # Spider抛出异常时调用该方法
    def process_spider_exception(self, response, exception, spider):
        # Called when a spider or process_spider_input() method
        # (from other spider middleware) raises an exception.

        # Should return either None or an iterable of Response, dict
        # or Item objects.
        pass

    # 处理入口url时调用
    def process_start_requests(self, start_requests, spider):
        # Called with the start requests of the spider, and works
        # similarly to the process_spider_output() method, except
        # that it doesn’t have a response associated.

        # Must return only requests (not items).
        for r in start_requests:
            yield r

    # spider 开启时调用该方法
    def spider_opened(self, spider):
        spider.logger.info('Spider opened: %s' % spider.name)


class JobDownloaderMiddleware(object):
    # Not all methods need to be defined. If a method is not defined,
    # scrapy acts as if the downloader middleware does not modify the
    # passed objects.

    # 创建下载器中间件时
    @classmethod
    def from_crawler(cls, crawler):
        # This method is used by Scrapy to create your spiders.
        s = cls()
        crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
        return s

    # 发送任何请求到下载器时
    def process_request(self, request, spider):
        # Called for each request that goes through the downloader
        # middleware.

        # Must either:
        # - return None: continue processing this request
        # - or return a Response object
        # - or return a Request object
        # - or raise IgnoreRequest: process_exception() methods of
        #   installed downloader middleware will be called
        return None

    # 下载器构建响应对象时调用该方法
    def process_response(self, request, response, spider):
        # Called with the response returned from the downloader.

        # Must either;
        # - return a Response object
        # - return a Request object
        # - or raise IgnoreRequest
        return response

    # 抛出异常时
    def process_exception(self, request, exception, spider):
        # Called when a download handler or a process_request()
        # (from other downloader middleware) raises an exception.

        # Must either:
        # - return None: continue processing this exception
        # - return a Response object: stops process_exception() chain
        # - return a Request object: stops process_exception() chain
        pass

    # spider开启时调用
    def spider_opened(self, spider):
        spider.logger.info('Spider opened: %s' % spider.name)
```

- Scrapy-redis

```
REDIS_HOST='' # redis服务器的IP
REDIS_PORT=8000 # redis服务器的端口
SCHEDULER='scrapy_redis.scheduler.Scheduler'
DUPEFILTER_CLASS='scrapy_redis.dupefilter.RFPDupeFilter'

要提前安装 pip install scrapy-redis
```

- 注意：

使用scrapy-dedis 仅仅是给调度器配置了redis，pipeline中的存储依旧使用mysql，切忌混淆