**import** re  
**import** random  
**import** hashlib  
  
  
**def** RandomStr(randLength=5):  
 str = **''** chars = **'AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz0123456789'** length = len(chars) - 1  
 **for** i **in** range(randLength):  
 str += chars[random.randint(0, length)]  
 **return** str  
  
  
**def** GetMd5(url):  
 **if** isinstance(url, str):  
 url = url.encode(**"utf-8"**)  
 m = hashlib.md5()  
 m.update(url)  
 **return** m.hexdigest()  
  
**def** UrlChange(url):  
 url = url.split(**"://"**)  
 url = url[1] **if** len(url) > 1 **else** url[0]  
 url = url.replace(**"/"**, ConstValue[**'BACKLASH'**])  
 url = url.replace(**"?"**, ConstValue[**'QUEST'**])  
 **return** url  
  
**def** UrlConvert(url):  
 url=UrlChange(url)  
 randStr = RandomStr()  
 **return** url + randStr  
  
  
**def** ChangeLabelText(str):  
 **return** re.sub(**'[\n\r\t ]'**, **''**, str)  
  
  
**def** GetLinkText(xpathLink):  
 **return** ChangeLabelText(**""**.join(xpathLink.xpath(**'./text()'**).getall())) **or** \  
 ChangeLabelText(xpathLink.attrib.get(**'title'**, **''**)) **or** ChangeLabelText(  
 **""**.join(xpathLink.xpath(**'.//text()'**).getall()))  
  
  
**def** WebUrlType(url):  
 RE\_NOT\_VISIT\_MATCH = **".\*/page\..\*|.\*guestbook\.asp.\*|.\*news\.hfut\.edu\.cn/info/\d\*/\d\*\..{0,5}$|.\*/show-\d+-\d+-\d+\..{0,5}$|.\*/\d+-[-\d]+\d+$|.\*/print-[-\d]+\d+\..{0,5}$"** INDEXURL = **'.\*main.{0,6}$|.\*index.{0,6}$|.\*cn/?$'** LISTURL = **".\*list.\*|.\*index.\*|.\*main.\*"  
 if not** re.match(RE\_NOT\_VISIT\_MATCH, url) == **None**:  
 **return** WEBPAGETYPE[**'CONTENTPAGE'**]  
 **if not** re.match(INDEXURL, url) == **None**:  
 **return** WEBPAGETYPE[**'INDEXPAGE'**]  
 **if not** re.match(LISTURL, url) == **None**:  
 **return** WEBPAGETYPE[**'LISTPAGE'**]  
 **return** WEBPAGETYPE[**'NONE'**]  
  
  
**def** WebPageType(response):  
 **try**:  
 urlType = WebUrlType(response.url)  
 **if** urlType == WEBPAGETYPE[**'CONTENTPAGE'**]:  
 **return** WEBPAGETYPE[**'CONTENTPAGE'**]  
 **if** urlType == WEBPAGETYPE[**'INDEXPAGE'**]:  
 **return** WEBPAGETYPE[**'INDEXPAGE'**]  
 linkList = []  
 otherList = []  
 **for** item **in** response.xpath(**'//body//a'**):  
 linkText = GetLinkText(item)  
 **if** linkText:  
 linkList.append(linkText)  
 otherLabelList = response.xpath(  
 **'//body//\*[name(.)!="a" and name(.)!="script" and name(.)!="style" and name(.)!="img"]'**)  
 **for** item **in** otherLabelList:  
 **if not** len(item.xpath(**"./ancestor::a"**)):  
 otherText = ChangeLabelText(**""**.join(item.xpath(**'./text()'**).getall()))  
 **if** otherText:  
 otherList.append(otherText)  
 linkLen = len(linkList)  
 otherLen = len(otherList)  
 validLinkText = **""**.join(linkList)  
 validOtherText = **""**.join(otherList)  
 linkTextLen = len(validLinkText)  
 otherTextLen = len(validOtherText)  
 signCount = len(re.findall(**r'。'**, validOtherText))  
 **if not** linkLen **or not** linkTextLen:  
 **return** WEBPAGETYPE[**'CONTENTPAGE'**]  
 **elif not** otherLen **or not** otherTextLen:  
 **return** WEBPAGETYPE[**'INDEXPAGE'**]  
 **else**:  
 **if** urlType == WEBPAGETYPE[**'LISTPAGE'**]:  
 **return** WEBPAGETYPE[  
 **'CONTENTPAGE'**] **if** linkLen / otherTextLen <= 0.1 **and** signCount >= 5 **and** otherTextLen / linkTextLen >= 1.5 **else** \  
 WEBPAGETYPE[**'INDEXPAGE'**]  
 **return** WEBPAGETYPE[  
 **'CONTENTPAGE'**] **if** linkLen / otherTextLen <= 0.3 **and** signCount >= 3 **and** otherTextLen / linkTextLen >= 1 **else** \  
 WEBPAGETYPE[**'INDEXPAGE'**]  
 **except**:  
 **return** WEBPAGETYPE[**'NONE'**]  
  
  
WEBPAGETYPE = {  
 **'INDEXPAGE'**: **'index'**,  
 **'CONTENTPAGE'**: **'content'**,  
 **'LISTPAGE'**: **'list'**,  
 **'NONE'**: **'none'**}  
  
ConstValue = {  
 **'MAX\_HEIGHT'**: 800,  
 **'MAX\_WIDTH'**: 1100,  
 **'imageUrl'**: **'imageUrl'**,  
 **'RE\_SORT\_MATCH'**: **'.\*//(.\*\.hfut\.edu\.cn).\*'**,  
 **'BACKLASH'**: **'+Z\_\*-'**,  
 **'QUEST'**: **'\*+Y\_-'**}  
EXTRACT\_IFNO = {  
 **'USELESS\_TAG'**: [**'style'**, **'script'**, **'link'**, **'video'**, **'iframe'**, **'source'**, **'picture'**, **'header'**, **'blockquote'**,  
 **'noscript'**, **'br'**, **'None'**, **'img'**, **'meta'**],  
 **'LINK'**: **'a'**,  
 **'CONTENT'**: **'p'**,  
 **'NONE'**: **'None'**,  
 **'SIGNSET'**: **r'''！|，|。|？|、|；|：|“|”|‘|’|《|》|（|）|,|\.|\?|:|;|'|"|!|\(|\)'''**}

*# -\*- coding: utf-8 -\*-***from** SearchSpider.Modules.CommonModule **import** EXTRACT\_IFNO  
**from** lxml.html **import** etree  
**import** re  
**import** numpy **as** np  
**from** datetime **import** datetime  
  
**class** ExtractInfo:  
 **def** \_\_init\_\_(self):  
 self.titleText = **''** self.time = **''** self.contentText = **''** self.nodeList = []  
  
 **def** IsValidYear(self,year):  
 **try**:  
 year=int(year)  
 **except**:  
 year=9999  
 **return** 2000<=year<=datetime.today().year  
  
 **def** ExtractIndexDate(self,htmlText):  
 regSub = **"([1-2][0-9]{3})[^0-9]{1,3}?([0-1]?[0-9])[^0-9]{0,3}?([0-3]?[0-9]?)"** reResSub = re.findall(regSub, htmlText, flags=re.S)  
 maxYear=**""  
 for** item **in** reResSub[:-1]:  
 **if** self.IsValidYear(item[0]):  
 **if** maxYear **and** item[0]>maxYear:  
 self.time=item[0]+**"-01-01"  
 elif not** maxYear:  
 maxYear=item[0]  
 self.time = item[0]+**"-01-01"  
 if** len(reResSub)==1 **and** self.IsValidYear(reResSub[0][0]):  
 self.time=reResSub[0][0]+**"-01-01"  
 def** ExtactProcess(self, htmlText, isIndex):  
 htmlText = re.sub(**r'</?span.\*?>|&.{1,6}?;|</?tr.\*?>|</?td.\*?>'**, **''**, htmlText, flags=re.S)  
 etreeObj = etree.HTML(htmlText)  
 self.titleText = self.ChangeNodeText(**""**.join(etreeObj.xpath(**'//title/text()'**)))  
 etree.strip\_elements(etreeObj, \*EXTRACT\_IFNO[**'USELESS\_TAG'**],with\_tail=**False**)  
 **if** isIndex:  
 self.ExtractIndexPro(etreeObj)  
 self.ExtractIndexDate(**""**.join(etreeObj.xpath(**'//text()'**)))  
 **else**:  
 self.ExtractPagePro(etreeObj)  
  
 **def** ExtractIndexPro(self, etreeObj):  
 otherAvg, otherNodes = self.GetAvgLen(etreeObj.xpath(**'//body//\*[name(.)!="a" and count(\*)=0]'**))  
 **for** item, otherLen **in** otherNodes:  
 parentNode = item.getparent()  
 **if** (otherLen < 0.5 \* otherAvg **or** otherLen < 5) **and** parentNode **is not None**:  
 parentNode.remove(item)  
 linkAvg, linkNodes = self.GetAvgLen(etreeObj.xpath(**'//body//a'**))  
 **for** item, linkLen **in** linkNodes:  
 parentNode = item.getparent()  
 **if** (linkLen < linkAvg **or** linkLen < 5) **and** parentNode **is not None**:  
 parentNode.remove(item)  
 self.contentText = self.ChangeNodeText(**""**.join(etreeObj.xpath(**'//body//text()'**)))  
  
 **def** GetAvgLen(self, NodesList):  
 count = 0  
 sum = 0  
 Nodes = []  
 **for** item **in** NodesList:  
 itemText = item.xpath(**'.//text()'**)  
 linkLen = len(self.ChangeNodeText(**""**.join(itemText)).strip())  
 Nodes.append([item, linkLen])  
 **if** linkLen > 4:  
 count += 1  
 sum += linkLen  
 **return** sum / count **if** count **else** 0, Nodes  
  
 **def** ExtractPagePro(self, etreeObj):  
 self.ExtractPage(etreeObj.xpath(**'//body/\*'**))  
 score\_list = [x[7] **for** x **in** self.nodeList]  
 std=1  
 **if** len(self.nodeList)>1:  
 std = np.std(score\_list, ddof=1)  
 maxItem = **None** maxScore = **None** maxIndex = 0  
 mayDate=**""** replaceDate=**""  
 for** index, item **in** enumerate(self.nodeList):  
 SBDi = (item[1] - item[3]) / (item[5] + 1) **if** item[1] - item[3] > 0 **else** 1  
 score = np.log(std) \* item[7] \* np.log10(item[6] + 2) \* np.log(SBDi) \* np.sqrt(item[5])  
 **if** item[8]:  
 replaceDate=item[8]  
 **if not** maxScore **or** maxScore < score:  
 maxScore = score  
 maxItem = item  
 maxIndex = index  
 **if** replaceDate:  
 mayDate=replaceDate  
 replaceDate=**""  
  
 if** maxItem :  
 self.contentText = self.nodeList[maxIndex][0]  
 dateIndex=maxIndex  
 **if not** self.time:  
 notConentIndex = maxIndex - maxItem[2]  
 dateIndex = notConentIndex  
 **while** dateIndex >= 0 **and** notConentIndex - dateIndex <= 5:  
 **if** self.nodeList[dateIndex][8]:  
 self.time = self.nodeList[dateIndex][8]  
 **break** dateIndex -= 1  
 **if not** self.time:  
 dateIndex = maxIndex  
 **while** dateIndex > notConentIndex:  
 **if** self.nodeList[dateIndex][8]:  
 self.time = self.nodeList[dateIndex][8]  
 **break** dateIndex -= 1  
 **if not** self.time:  
 self.time=mayDate  
 **if not** self.titleText:  
 titleIndex = dateIndex  
 **while** titleIndex >= 0 **and** dateIndex - titleIndex <= 15:  
 **if** self.nodeList[titleIndex][9]:  
 self.titleText = self.nodeList[titleIndex][9]  
 **break** titleIndex -= 1  
  
 **def** ExtractPage(self, SelectInfo):  
 elemList = []  
 **for** item **in** SelectInfo:  
 **try**:  
 tag = item.tag  
 **except**:  
 tag = EXTRACT\_IFNO[**'NONE'**]  
 **if not** tag **in** EXTRACT\_IFNO[**'USELESS\_TAG'**]:  
 nodeText = self.ChangeNodeText(**""**.join(item.xpath(**'.//text()'**))).strip()  
 **if** nodeText:  
 childList = self.ExtractPage(item.xpath(**'./\*'**))  
 Ti, TGi, LTi, LTGi, Sbi, PNum,calTGi = self.GetNodeInfo(nodeText, tag)  
 subTi = subTGi = subLTi = subLTGi = subPNum =subCalTGi= 0  
 **for** childItem **in** childList:  
 subTi += childItem[1]  
 subTGi += childItem[2]  
 subLTi += childItem[3]  
 subLTGi += childItem[4]  
 subPNum += childItem[6]  
 subCalTGi+=childItem[8]  
 LTi = (LTi - subTi **if** LTi > 0 **else** LTi) + subLTi  
 TGi += subTGi  
 LTGi += subLTGi  
 PNum += subPNum  
 calTGi+=subCalTGi  
 a = 1 / LTGi **if** LTGi **else** 1  
 TDi = ((Ti - LTi) / (calTGi - LTGi)) \* a **if** calTGi - LTGi > 0 **else** 0  
 elemList.append([nodeText, Ti, TGi, LTi, LTGi, Sbi, PNum, TDi,calTGi])  
 self.nodeList.append([nodeText, Ti, TGi, LTi, LTGi, Sbi, PNum, TDi, self.ExtractDate(nodeText),  
 self.ExtractTitle(nodeText, tag)])  
 **return** elemList  
  
 **def** GetNodeInfo(self, nodeText, tag):  
 Sbi = len(re.findall(EXTRACT\_IFNO[**'SIGNSET'**], nodeText, flags=re.S))  
 Ti = len(nodeText) \*\* 1.5  
 TGi = 1  
 calTGi=1  
 LTi = 0  
 LTGi = 0  
 PNum = 0  
 **if** tag == EXTRACT\_IFNO[**'LINK'**]:  
 LTi = Ti  
 LTGi = TGi  
 **elif** tag == EXTRACT\_IFNO[**'CONTENT'**]:  
 PNum = 1  
 **if** tag!=EXTRACT\_IFNO[**'LINK'**] **and** len(nodeText)<10:  
 calTGi=0  
 **if** (**'Copyright' in** nodeText **or '版权所有' in** nodeText) **and '©' in** nodeText:  
 Ti\*\*=0.5  
 calTGi=1  
 **return** Ti, TGi, LTi, LTGi, Sbi, PNum,calTGi  
  
 **def** ChangeNodeText(self, str):  
 subStr = re.sub(**'[\n\r\t]'**, **''**, str, flags=re.S)  
 **return** re.sub(**' +'**, **' '**, subStr, flags=re.S)  
  
 **def** ExtractDate(self, str):  
 reg = **"((发布|更新|编辑)?(时间|日期).{0,3})?([1-2][0-9]{3})[^0-9]{1,3}?([0-1]?[0-9])[^0-9]{1,3}?([0-3]?[0-9])"** probDate = **''  
 if not** self.time:  
 reRes = re.findall(reg, str, flags=re.S)  
 **for** item **in** reRes:  
 **if** self.IsValidYear(item[3]):  
 **if** item[1]:  
 self.time=**"-"**.join(item[3:])  
 **break  
 elif not** probDate :  
 probDate=**"-"**.join(item[3:])  
 **if not** self.time:  
 regSub=**"^日期.{0,3}?([1-2][0-9]{3})[^0-9]{1,3}?([0-1]?[0-9])[^0-9]{1,3}?([0-3]?[0-9])"** reResSub=re.findall(regSub,str, flags=re.S)  
 **if** len(reResSub)==1:  
 **if** self.IsValidYear(reResSub[0][0]):  
 self.time = **"-"**.join(reResSub[0])  
 **return** probDate  
  
 **def** ExtractTitle(self, str, tag):  
 **return** str **if not** self.titleText **and** len(tag) **and** tag[0] == **'h' else ''  
  
 def** GetDate(self):  
 **return** self.time  
  
 **def** GetContent(self):  
 **return** self.contentText  
  
 **def** GetTitle(self):  
 **return** self.titleText

**from** scrapy **import** signals  
**from** scrapy.exceptions **import** NotConfigured  
  
**class** RecordProNumEx(object):  
 **def** \_\_init\_\_(self,fileName):  
 self.fileName = fileName  
  
 @classmethod  
 **def** from\_crawler(cls, crawler):  
 **if not** crawler.settings.getbool(**'MYEXT\_ENABLED'**):  
 **raise** NotConfigured  
 extens = cls(crawler.settings.get(**'CONFIG\_READ\_SETTING'**))  
 crawler.signals.connect(extens.spider\_opened, signal=signals.spider\_opened)  
 crawler.signals.connect(extens.spider\_closed, signal=signals.spider\_closed)  
 **return** extens  
  
  
 **def** WriteFile(self, typeStr):  
 **with** open(self.fileName, **'w'**) **as** f:  
 f.write(**"SPIDER:{0}"**.format(typeStr))  
  
 **def** spider\_opened(self, spider):  
 self.WriteFile(**"OPENED"**)  
 spider.logger.info(**"opened spider {}, Writing setting file:{}"**.format(spider.name, **"OPENED"**))  
  
 **def** spider\_closed(self, spider):  
 self.WriteFile(**"CLOSED"**)  
 spider.logger.info(**"closed spider {}, Writing setting file:{}"**.format(spider.name, **"CLOSED"**))

**from** scrapy **import** signals  
**from** scrapy.exceptions **import** NotConfigured  
  
  
**class** RedisSpiderClosedEx(object):  
  
 **def** \_\_init\_\_(self, idleNumber, crawler):  
 self.crawler = crawler  
 self.idleNumber = idleNumber  
 self.idleCount = 0  
  
 @classmethod  
 **def** from\_crawler(cls, crawler):  
  
 **if not** crawler.settings.getbool(**'MYEXT\_ENABLED'**):  
 **raise** NotConfigured  
  
 idleNumber = crawler.settings.getint(**'IDLE\_NUMBER'**, 360)  
  
 extens = cls(idleNumber, crawler)  
  
 crawler.signals.connect(extens.spider\_opened, signal=signals.spider\_opened)  
  
 crawler.signals.connect(extens.spider\_closed, signal=signals.spider\_closed)  
  
 crawler.signals.connect(extens.spider\_idle, signal=signals.spider\_idle)  
  
 **return** extens  
  
 **def** spider\_opened(self, spider):  
 spider.logger.info(**"opened spider {}, Allow waiting time:{} second"**.format(spider.name, self.idleNumber \* 5))  
  
 **def** spider\_closed(self, spider):  
 spider.logger.info(  
 **"closed spider {}, Waiting time exceeded {} second"**.format(spider.name, self.idleNumber \* 5))  
  
 **def** spider\_idle(self, spider):  
  
 **if** spider.server.exists(spider.redis\_key) **or** spider.server.exists(spider.redis\_request):  
 self.idleCount = 0  
 **else**:  
 self.idleCount += 1  
  
 **if** self.idleCount > self.idleNumber:  
 self.crawler.engine.close\_spider(spider, **'Waiting time exceeded'**)

*# -\*- coding: utf-8 -\*-***from** SearchSpider.Modules.CommonModule **import** WEBPAGETYPE  
**from** SearchSpider.items **import** SearchSpiderItem, SearchItemLoader  
**from** scrapy.spiders **import** Rule  
**from** scrapy.linkextractors **import** LinkExtractor  
**from** scrapy\_redis.spiders **import** RedisCrawlSpider  
**from** SearchSpider.Modules.ExtractContentInfo **import** ExtractInfo  
  
  
**class** HfutspiderSpider(RedisCrawlSpider):  
 name = **'HfutSpider'** allowed\_domains = [**'hfut.edu.cn'**]  
 redis\_key = **'HfutSpider:start\_urls'** redis\_request = **'HfutSpider:requests'** rules = (  
 Rule(LinkExtractor(allow=**r''**), callback=**'ParseItem'**, follow=**True**),  
 )  
  
 **def** ParseItem(self, response):  
 **if** response.meta[**"pageType"**] == WEBPAGETYPE[**'INDEXPAGE'**] **or** \  
 response.meta[**"pageType"**] == WEBPAGETYPE[**'CONTENTPAGE'**]:  
 isIndex=response.meta[**"pageType"**] == WEBPAGETYPE[**'INDEXPAGE'**]  
 itemLoader = SearchItemLoader(item=SearchSpiderItem(), response=response)  
 extract = ExtractInfo()  
 extract.ExtactProcess(response.text,isIndex)  
 itemLoader.add\_value(**"url"**, response.url)  
 itemLoader.add\_value(**"urlOrigin"**, response.url)  
 **if** response.meta[**"pageType"**]== WEBPAGETYPE[**'CONTENTPAGE'**]:  
 itemLoader.add\_css(**"imageUrl"**, **"img::attr(src)"**)  
 itemLoader.add\_value(**"title"**, extract.GetTitle())  
 itemLoader.add\_value(**"content"**, extract.GetContent())  
 itemLoader.add\_value(**"createdDate"**, extract.GetDate())  
 itemLoader.add\_value(**"isIndex"**, isIndex)  
 **return** itemLoader.load\_item()

*# -\*- coding: utf-8 -\*-***from** scrapy.loader **import** ItemLoader  
**from** scrapy.loader.processors **import** TakeFirst, Join, MapCompose  
**from** SearchSpider.Modules.CommonModule **import** ConstValue  
**import** re  
**import** scrapy  
  
  
**class** SearchItemLoader(ItemLoader):  
 default\_output\_processor = TakeFirst()  
  
  
**def** getDomain(value):  
 res = re.match(ConstValue[**'RE\_SORT\_MATCH'**], value)  
 **return "" if** res **is None else** res.group(1)  
  
  
**class** SearchSpiderItem(scrapy.Item):  
 url = scrapy.Field()  
 title = scrapy.Field()  
 content = scrapy.Field()  
 isIndex=scrapy.Field()  
 imageResult=scrapy.Field()  
 urlOrigin = scrapy.Field(  
 input\_processor=MapCompose(getDomain)  
 )  
 createdDate = scrapy.Field()  
 imageUrl = scrapy.Field(  
 output\_processor=Join(**","**)  
 )

*# -\*- coding: utf-8 -\*-***from** scrapy **import** signals  
**from** scrapy.utils.misc **import** load\_object  
**from** scrapy\_redis **import** defaults, connection  
**from** SearchSpider.Modules.CommonModule **import** WebPageType, WEBPAGETYPE  
  
  
**class** MyMiddleware(object):  
 **def** \_\_init\_\_(self, notVisit, notDownload):  
 self.notVisit = notVisit  
 self.notDownload = notDownload  
  
 @classmethod  
 **def** from\_crawler(cls, crawler):  
 notVisit = load\_object(defaults.SCHEDULER\_NOT\_VISIT\_CLASS).from\_crawler(crawler)  
 notDownload = load\_object(defaults.SCHEDULER\_NOT\_DOWNLOAD\_CLASS).from\_crawler(crawler)  
 **return** cls(notVisit, notDownload)  
  
 **def** process\_response(self, request, response, spider):  
 **if** response.status == 200 **and** request.meta.get(**'IsWeb'**,**False**):  
 pageType = WebPageType(response)  
 isNotDown=self.notDownload.is\_exist(request)  
 **if** isNotDown **and** pageType==WEBPAGETYPE[**'CONTENTPAGE'**]:  
 pageType=WEBPAGETYPE[**'INDEXPAGE'**]  
 **if** pageType == WEBPAGETYPE[**'INDEXPAGE'**]:  
 request.meta[**'IsNew'**] = **not** self.notDownload.request\_seen(request)  
 **elif** pageType == WEBPAGETYPE[**'CONTENTPAGE'**]:  
 self.notVisit.request\_seen(request)  
 request.meta[**'pageType'**] = pageType  
 **else**:  
 request.meta[**'pageType'**] = WEBPAGETYPE[**'NONE'**]  
 **return** response  
  
  
**class** SearchspiderSpiderMiddleware(object):

@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  
  
 **def** process\_spider\_input(self, response, spider):

**return None  
  
 def** process\_spider\_output(self, response, result, spider):

**for** i **in** result:  
 **yield** i  
  
 **def** process\_spider\_exception(self, response, exception, spider):

**pass  
  
 def** process\_start\_requests(self, start\_requests, spider):

**for** r **in** start\_requests:  
 **yield** r  
  
 **def** spider\_opened(self, spider):  
 spider.logger.info(**'Spider opened: %s'** % spider.name)  
  
  
**class** SearchspiderDownloaderMiddleware(object):

@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  
  
 **def** process\_request(self, request, spider):

**return None  
  
 def** process\_response(self, request, response, spider):

**return** response  
  
 **def** process\_exception(self, request, exception, spider):

**pass  
  
 def** spider\_opened(self, spider):  
 spider.logger.info(**'Spider opened: %s'** % spider.name)

*# -\*- coding: utf-8 -\*-***from** urllib **import** parse  
**from** scrapy.http **import** Request  
**from** scrapy.pipelines.images **import** ImagesPipeline, ImageException  
**from** SearchSpider.Modules.CommonModule **import** UrlConvert, ConstValue,GetMd5,UrlChange  
**from** PIL **import** Image  
**from** scrapy\_redis.pipelines **import** RedisPipeline  
**import** six  
  
**try**:  
 **from** cStringIO **import** StringIO **as** BytesIO  
**except** ImportError:  
 **from** io **import** BytesIO  
  
  
**class** SpiderTOEsPipeline(RedisPipeline):  
 **pass  
  
  
class** ImageDealPipeline(ImagesPipeline):  
 **def** \_\_init\_\_(self, store\_uri, download\_func=**None**, settings=**None**):  
 super(ImageDealPipeline, self).\_\_init\_\_(store\_uri, settings=settings,  
 download\_func=download\_func)  
 self.imgNameDict = {}  
  
 **def** get\_media\_requests(self, item, info):  
 **if** ConstValue[**"imageUrl"**] **in** item:  
 urls = item[**"imageUrl"**].split(**','**)  
 **return** [Request(parse.urljoin(item[**"url"**], x), meta={**"domain"**: item[**"url"**]}) **for** x **in** urls]  
  
 **def** get\_images(self, response, request, info):  
 path = self.file\_path(request, response=response, info=info)  
 origImage = Image.open(BytesIO(response.body))  
 width, height = origImage.size  
 **if** width < self.min\_width **or** width > ConstValue[**'MAX\_WIDTH'**] **or** \  
 height < self.min\_height **or** height > ConstValue[**'MAX\_HEIGHT'**]:  
 urlMd5=GetMd5(request.url)  
 **if** self.imgNameDict.get(urlMd5):  
 **del** self.imgNameDict[urlMd5]  
 **raise** ImageException(**"Image not is standard size"**)  
  
 image, buf = self.convert\_image(origImage)  
 **yield** path, image, buf  
  
 **for** thumb\_id, size **in** six.iteritems(self.thumbs):  
 thumb\_path = self.thumb\_path(request, thumb\_id, response=response, info=info)  
 thumb\_image, thumb\_buf = self.convert\_image(image, size)  
 **yield** thumb\_path, thumb\_image, thumb\_buf  
  
 **def** item\_completed(self, results, item, info):  
 urlCom=UrlChange(item[**'url'**] **if 'url' in** item **else ''**)  
 item[**'imageResult'**] =**","**.join([x[**'path'**] **for** ok, x **in** results **if** ok **and** x[**'path'**][:-9]==urlCom])  
 **return** item  
  
 **def** file\_path(self, request, response=**None**, info=**None**):  
 urlMd5=GetMd5(request.url)  
 imgName = self.imgNameDict.get(urlMd5)  
 **if not** imgName:  
 imgName = **'%s.jpg'** % (UrlConvert(request.meta.get(**"domain"**, **""**)))  
 self.imgNameDict[urlMd5] = imgName  
 **return** imgName

*# -\*- coding: utf-8 -\*-***import** os  
  
BOT\_NAME = **'SearchSpider'**SPIDER\_MODULES = [**'SearchSpider.spiders'**]  
NEWSPIDER\_MODULE = **'SearchSpider.spiders'***# Obey robots.txt rules*ROBOTSTXT\_OBEY = **False**LOG\_LEVEL = **'INFO'**LOG\_FILE = **'exception.log'**COOKIES\_ENABLED = **False**RETRY\_ENABLED = **False**

DOWNLOADER\_MIDDLEWARES = {  
 **'SearchSpider.middlewares.MyMiddleware'**: 2  
}  
*# Enable or disable extensions  
# See https://doc.scrapy.org/en/latest/topics/extensions.html*EXTENSIONS = {  
 *# 'scrapy.extensions.telnet.TelnetConsole': None,* **'SearchSpider.Modules.RecordProNumExtension.RecordProNumEx'**: 500,  
 **'SearchSpider.Modules.RedisSpiderClosedExension.RedisSpiderClosedEx'**: 300  
}  
  
*# Configure item pipelines  
# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html*ITEM\_PIPELINES = {  
 *# 'SearchSpider.pipelines.SearchspiderPipeline': 300,* **'SearchSpider.pipelines.ImageDealPipeline'**: 1,  
 **'SearchSpider.pipelines.SpiderTOEsPipeline'**: 2,  
}  
  
MYEXT\_ENABLED = **True***# IDLE\_NUMBER = 360*SPIDER\_NAME = **"HfutSpider"**SCHEDULER = **"scrapy\_redis.scheduler.Scheduler"**DUPEFILTER\_CLASS = **"scrapy\_redis.dupefilter.RFPDupeFilter"**USER\_AGENT = **"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0"**IMAGES\_MIN\_HEIGHT = **"200"**IMAGES\_MIN\_WIDTH = **"350"**IMAGES\_URLS\_FIELD = **"imageUrl"**projectDir = os.path.abspath(os.path.dirname(os.path.dirname(os.path.dirname(\_\_file\_\_))))  
CONFIG\_READ\_SETTING = os.path.join(projectDir, **'ProcessSetting'**)  
IMAGES\_STORE = os.path.join(projectDir, **'Images'**)  
**from** scrapy.cmdline **import** execute  
  
**import** sys  
**import** os  
  
sys.path.append(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
execute([**"scrapy"**, **"crawl"**, **"HfutSpider"**])

**from** scrapy.dupefilters **import** BaseDupeFilter  
**from** scrapy.utils.request **import** request\_fingerprint  
  
**from** scrapy\_redis **import** get\_redis\_from\_settings  
**from** scrapy\_redis.pipelines **import** default\_serialize  
**from** . **import** defaults  
  
  
**class** RedisNotDownloadUrl(BaseDupeFilter):  
 **def** \_\_init\_\_(self, server,  
 key=defaults.NOT\_DOWNLOAD\_URL\_KEY,  
 serialize\_func=default\_serialize):  
 self.server = server  
 self.key = key  
 self.serialize\_func = serialize\_func  
  
 **def** request\_seen(self, request):  
 fp = request\_fingerprint(request)  
 added = self.server.sadd(self.key, fp)  
 **return** added == 0  
  
 **def** is\_exist(self, request):  
 fp = request\_fingerprint(request)  
 **return** self.server.sismember(self.key, fp)  
  
 @classmethod  
 **def** from\_spider(cls, spider):  
 settings = spider.settings  
 server = get\_redis\_from\_settings(settings)  
 not\_download\_key = settings.get(**"SCHEDULER\_NOT\_DOWNLOAD\_KEY"**, defaults.SCHEDULER\_NOT\_DOWNLOAD\_KEY)  
 key = not\_download\_key % {**'spider'**: spider.name}  
 **return** cls(server, key=key)  
  
 @classmethod  
 **def** from\_crawler(cls, crawler):  
 **return** cls.from\_settings(crawler.settings)  
  
 @classmethod  
 **def** from\_settings(cls, settings):  
 server = get\_redis\_from\_settings(settings)  
 key = **'%s:notdownloadurl'** % settings.get(**"SPIDER\_NAME"**, **"HfutSpider"**)  
 **return** cls(server, key=key)

**from** scrapy.dupefilters **import** BaseDupeFilter  
**from** scrapy.utils.request **import** request\_fingerprint  
  
**from** scrapy\_redis **import** get\_redis\_from\_settings  
**from** scrapy\_redis.pipelines **import** default\_serialize  
**from** . **import** defaults  
  
  
**class** RedisNotVisitUrl(BaseDupeFilter):  
 **def** \_\_init\_\_(self, server,  
 key=defaults.NOT\_VISIT\_URL\_KEY,  
 serialize\_func=default\_serialize):  
 self.server = server  
 self.key = key  
 self.serialize\_func = serialize\_func  
  
 **def** request\_seen(self, request):  
 fp = request\_fingerprint(request)  
 added = self.server.sadd(self.key, fp)  
 **return** added == 0  
  
 **def** is\_exist(self, request):  
 fp = request\_fingerprint(request)  
 **return** self.server.sismember(self.key, fp)  
  
 @classmethod  
 **def** from\_spider(cls, spider):  
 settings = spider.settings  
 server = get\_redis\_from\_settings(settings)  
 not\_visit\_key = settings.get(**"SCHEDULER\_NOT\_VISIT\_KEY"**, defaults.SCHEDULER\_NOT\_VISIT\_KEY)  
 key = not\_visit\_key % {**'spider'**: spider.name}  
 **return** cls(server, key=key)  
  
 @classmethod  
 **def** from\_crawler(cls, crawler):  
 **return** cls.from\_settings(crawler.settings)  
  
 @classmethod  
 **def** from\_settings(cls, settings):  
 server = get\_redis\_from\_settings(settings)  
 key = **'%s:notvisiturl'** % settings.get(**"SPIDER\_NAME"**, **"HfutSpider"**)  
 **return** cls(server, key=key)

**import** re  
**import** six  
  
  
**def** bytes\_to\_str(s, encoding=**'utf-8'**):  
 *"""Returns a str if a bytes object is given."""* **if** six.PY3 **and** isinstance(s, bytes):  
 **return** s.decode(encoding)  
 **return** s  
  
  
**def** jump\_url(url):  
 RE\_NOT\_VISIT\_MATCH = **".\*news\.hfut\.edu\.cn/yzhy\.jsp\?urltype=tree.TreeTempUrl.\*"** res = re.match(RE\_NOT\_VISIT\_MATCH, url)  
 **return not** res == **None**

**import** logging  
**from** CommonFile **import** LOGGER\_PARAMS  
  
**class** AllLogger:  
 @staticmethod  
 **def** InitPicLog():  
 logFile = logging.FileHandler(LOGGER\_PARAMS[**'PIC\_FILE'**], **'w'**)  
 dataLog = logging.Logger(LOGGER\_PARAMS[**'PIC\_LOG'**], level=logging.INFO)  
 dataLog.addHandler(logFile)  
 **return** dataLog  
  
 @staticmethod  
 **def** InitErrorLog():  
 logFile = logging.FileHandler(LOGGER\_PARAMS[**'ERROR\_FILE'**], **'a'**)  
 fat = logging.Formatter(fmt=**"%(asctime)s - %(name)s - %(levelname)s - %(module)s - %(message)s"**)  
 logFile.setFormatter(fat)  
 dataLog = logging.Logger(LOGGER\_PARAMS[**'ERROR\_LOG'**], level=logging.ERROR)  
 dataLog.addHandler(logFile)  
 **return** dataLog

**import** os  
**import** hashlib  
baseDir = os.path.abspath(os.path.dirname(os.path.dirname(\_\_file\_\_)))  
webPicDir = os.path.join(baseDir, **'HfutSearch/static'**)  
  
VGG\_PARAMS = {  
 **'weight'**: **'imagenet'**,  
 **'inputShape'**: (224, 224, 3),  
 **'includeTop'**: **False**,  
 **'pooling'**: **'max'**,  
}  
  
configIfno = {  
 **'settingFile'**: os.path.join(baseDir, **"ProcessSetting"**),  
 **'tempImg'**: os.path.join(baseDir, **"Images"**) + **'/\*.jpg'**,  
 **'OPENED'**: **'OPENED'**,  
 **'CLOSED'**: **'CLOSED'**,  
 **'PIC\_MAX\_DEAL'**:500,  
 **'PIC\_MIN\_INSERT'**: 300,  
 **'TEXT\_MIN\_INSERT'**: 1000,  
 **'INIT\_WAIT\_MAX\_TIME'**:30,  
 **'END\_WAIT\_MAX\_TIME'**:120  
}  
  
SAVE\_ES\_PARAMS = {  
 **'INDEX'**: **'hfut\_search'**,  
 **'TYPE'**: **'hfut\_type'**,  
 **'WEIGHT'**: 10,  
 **'reItemName'**: **'HfutSpider:items'**,  
 **'esConnect'**:{  
 **"host"**: **"localhost"**,  
 **"port"**: 16110  
 }  
  
}  
  
DEAL\_PARAMS = {  
 **'picSearTempPos'**: baseDir + **'/Images/\*.jpg'**,  
 **'picVGGDB'**: baseDir + **'/VGGGFatureCNN.h5'**,  
 **'picSearResPos'**: webPicDir + **'/Result/\*'**,  
 **'picSavePos'**: webPicDir + **'/Result'**,  
 **'featureDS'**: **'featureDS'**,  
 **'imgPathDS'**: **'imgPathDS'**,  
 **'imgNameFront'**: **'Result'**,  
 **'maxFileCount'**: 5000,  
  
 **'INDEX'**: **'hfut\_pic'**,  
 **'TYPE'**: **'hfut\_pic\_type'**,  
 **'esConnect'**: {  
 **"host"**: **"localhost"**,  
 **"port"**: 16110  
 }  
  
}  
  
LOGGER\_PARAMS = {  
 **'PIC\_LOG'**: **'PIC\_LOG'**,  
 **'ERROR\_LOG'**: **'ERROR\_LOG'**,  
 **'PIC\_FILE'**:**'./dataFile/picLog.log'**,  
 **'ERROR\_FILE'**:**'./dataFile/error.log'**}  
  
**def** GetMd5(url):  
 **if** isinstance(url, str):  
 url = url.encode(**"utf-8"**)  
 m = hashlib.md5()  
 m.update(url)  
 **return** m.hexdigest()

**import** os  
**from** VGGNet **import** VGGNet  
**import** numpy **as** np  
**import** h5py  
**from** CommonFile **import** DEAL\_PARAMS  
**import** shutil  
**from** pathlib **import** Path  
**import** time  
**import** uuid  
**from** elasticsearch **import** helpers  
**from** datetime **import** datetime  
**from** elasticsearch\_dsl.connections **import** connections  
  
**class** PictureDeal:  
  
 **def** \_\_init\_\_(self, logger, maxDicFile, \*\*kwargs):  
 self.picSearResPos = kwargs[**'picSearResPos'**]  
 self.maxFileCount = kwargs[**'maxFileCount'**]  
 self.picVGGDB = kwargs[**'picVGGDB'**]  
 self.picSearTempPos = kwargs[**'picSearTempPos'**]  
 params = kwargs[**'esConnect'**]  
 self.es = connections.create\_connection(\*\*params)  
 self.esIndex = kwargs[**'INDEX'**]  
 self.esType = kwargs[**'TYPE'**]  
 self.featureDS = kwargs[**'featureDS'**]  
 self.imgPathDS = kwargs[**'imgPathDS'**]  
 self.imgNameFront = kwargs[**'imgNameFront'**]  
 self.picSavePos = kwargs[**'picSavePos'**]  
 self.maxTargetDic = maxDicFile[0]  
 self.remainFileCout = self.maxFileCount - maxDicFile[1]  
 self.model = VGGNet.CreateVGGNet()  
 self.picLog = logger  
 self.picDict={}  
  
 **def** CalculDeal(self, imgList):  
  
 **if** self.remainFileCout <= 0:  
 self.maxTargetDic += 1  
 self.remainFileCout = self.maxFileCount  
 targetDicPath = os.path.join(self.picSavePos, str(self.maxTargetDic))  
 targetDic = Path(targetDicPath)  
 h5FeatureDS = self.featureDS + str(self.maxTargetDic)  
 h5ImgPathDS = self.imgPathDS + str(self.maxTargetDic)  
 **if not** targetDic.is\_dir():  
 os.mkdir(targetDicPath)  
 featList, desImgList,picSet = self.OutputFeature(imgList, targetDicPath,  
 os.path.join(self.imgNameFront, str(self.maxTargetDic)))  
 addFileCount = self.SaveExtractInfo(featList, desImgList, h5FeatureDS, h5ImgPathDS)  
 self.PictureSaveES()  
 self.remainFileCout -= addFileCount  
 **return** picSet  
 **def** OutputFeature(self, imgList, targetDic, maxDic):  
 picSet=set()  
 featList = []  
 desImgList = []  
 **for** srcImgPath **in** imgList:  
 **try**:  
 normFeat, imgSize = self.model.VGGExtractFeat(srcImgPath)  
 imgName = os.path.split(srcImgPath)[1]  
 desImgPath = os.path.join(targetDic, imgName)  
 strSize = str(imgSize[0]) + **'X'** + str(imgSize[1])  
 imgSavePath = os.path.join(maxDic, imgName)  
 imgSavePath = imgSavePath + strSize  
 shutil.move(srcImgPath, desImgPath)  
 featList.append(normFeat)  
 desImgList.append(imgSavePath)  
 picSet.add(imgName)  
 self.picDict[imgName]=imgSavePath  
 **except**:  
 self.picLog.error(srcImgPath + **':photo exception！'**)  
 **return** featList, desImgList,picSet  
  
 **def** OpenH5py(self):  
 h5File = h5py.File(self.picVGGDB)  
 **while** h5File.mode != **'r+'**:  
 time.sleep(10)  
 h5File.close()  
 h5File = h5py.File(self.picVGGDB)  
 **return** h5File  
  
 **def** SaveExtractInfo(self, featList, desImgList, h5FeatureDS, h5ImgPathDS):  
 feats = np.array(featList)  
 shapeLen = list(feats.shape)  
 h5File = self.OpenH5py()  
 **try**:  
 **if not** h5ImgPathDS **in** h5File **and not** h5FeatureDS **in** h5File:  
 shapeLen[0] = **None** strDataSet=h5File.create\_dataset(h5ImgPathDS, (len(desImgList),), maxshape=tuple([**None**, ]), dtype=h5py.string\_dtype())  
 strDataSet[:]=desImgList  
 h5File.create\_dataset(h5FeatureDS, data=feats, maxshape=tuple(shapeLen))  
 **else**:  
 datasetFeat = h5File[h5FeatureDS]  
 datasetImg = h5File[h5ImgPathDS]  
 oldLen = datasetImg.shape[0]  
 newLen = oldLen + shapeLen[0]  
 shapeLen[0] = newLen  
 datasetFeat.resize(tuple(shapeLen))  
 datasetImg.resize(tuple([newLen, ]))  
 datasetFeat[oldLen:newLen] = feats  
 datasetImg[oldLen:newLen] = desImgList  
 **except**:  
 self.picLog.error(**'h5File open error!'**)  
 **finally**:  
 h5File.close()  
 **return** len(desImgList)  
  
 **def** PictureSaveES(self):  
 **try**:  
 elemList = []  
 **for** key, value **in** self.picDict.items():  
 esId = str(uuid.uuid4())  
 imgName = key  
 imgPath = value  
 saveDate = datetime.today()  
 elem = {  
 **"\_index"**: self.esIndex,  
 **"\_type"**: self.esType,  
 **"\_id"**: esId,  
 **"\_source"**: {  
 **"img\_name"**: imgName,  
 **"img\_path"**: imgPath,  
 **"save\_date"**: saveDate  
 }  
 }  
 elemList.append(elem)  
 helpers.bulk(self.es, elemList)  
 **except**:  
 self.picLog.error(**"saving to es error!"**)  
 **finally**:  
 self.picDict.clear()  
  
 @classmethod  
 **def** CreatePicDeal(cls, logger, maxDicFile):  
 **return** cls(logger, maxDicFile, \*\*DEAL\_PARAMS)

**from** DealPicture **import** PictureDeal  
**from** SaveEs **import** DealRedis  
**from** CommonFile **import** configIfno, SAVE\_ES\_PARAMS,DEAL\_PARAMS  
**from** scrapy\_redis **import** connection  
**from** scrapy.utils.project **import** get\_project\_settings  
**import** glob  
**import** sys  
**import** time  
**from** AllLoggerFile **import** AllLogger  
  
**def** ReadFile(path,logger):  
 **try**:  
 **with** open(path, **'r'**) **as** f:  
 statusSpider = f.read()  
 **return** statusSpider.split(**":"**)  
 **except**:  
 logger.error(**'read file error!'**)  
  
  
**def** getKeyLen(server):  
 **return** server.llen(SAVE\_ES\_PARAMS[**'reItemName'**])  
  
  
**def** getImgListLen():  
 imgList = glob.glob(configIfno[**'tempImg'**])  
 **return** imgList, len(imgList)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 settings = get\_project\_settings()  
 server = connection.from\_settings(settings)  
 logger=AllLogger.InitErrorLog()  
 **try**:  
 targetDics = [int(dic[len(DEAL\_PARAMS[**'picSearResPos'**][:-1]):]) **for** dic **in** glob.glob(DEAL\_PARAMS[**'picSearResPos'**])]  
 maxTargetDic= max(targetDics) **if** len(targetDics) **else** 0  
 maxDicFiles = glob.glob(DEAL\_PARAMS[**'picSearResPos'**][:-1] + str(maxTargetDic) + **'/\*'**)  
 FileCount = len(maxDicFiles)  
 **except**:  
 maxTargetDic = 0  
 FileCount=0  
 logger.error(**'catalogue name is wrong！'**)  
 picDeal = PictureDeal.CreatePicDeal(logger,(maxTargetDic,FileCount))  
 esSave = DealRedis.CreateDealRedis(server,logger)  
 statusSpider = ReadFile(configIfno[**'settingFile'**],logger)  
 count=0  
 **while not** getKeyLen(server) **and**(**not** statusSpider **or** len(statusSpider )!=2 **or** statusSpider[1] == configIfno[**'CLOSED'**]):  
 time.sleep(10)  
 count=count+1  
 **if** count==configIfno[**'INIT\_WAIT\_MAX\_TIME'**]:  
 logger.error(**'spider error running!'**)  
 sys.exit(0)  
 statusSpider = ReadFile(configIfno[**'settingFile'**],logger)  
 count=0  
 picSet=set()  
 **while True**:  
 imgList, imglen = getImgListLen()  
 **if** imglen > configIfno[**'PIC\_MIN\_INSERT'**]:  
 count=0  
 imgList=list(set(imgList)-picSet)  
 resPicSet=picDeal.CalculDeal(imgList[:configIfno[**'PIC\_MAX\_DEAL'**]])  
 picSet=set.union(picSet,resPicSet)  
 **if** getKeyLen(server) > configIfno[**'TEXT\_MIN\_INSERT'**]:  
 count=0  
 esSave.ReadRedis(configIfno[**'TEXT\_MIN\_INSERT'**])  
 imgList, imglen = getImgListLen()  
 redisCount=getKeyLen(server)  
 **if** imglen <= configIfno[**'PIC\_MIN\_INSERT'**] **and** redisCount <= configIfno[**'TEXT\_MIN\_INSERT'**]:  
 statusSpider = ReadFile(configIfno[**'settingFile'**],logger)  
 **if** count==configIfno[**'END\_WAIT\_MAX\_TIME'**] **or** \  
 (statusSpider **and** len(statusSpider) == 2 **and** statusSpider[1] == configIfno[**'CLOSED'**]):  
 esSave.ReadRemain()  
 imgList, imglen = getImgListLen()  
 imgList = list(set(imgList) - picSet)  
 **if** len(imgList):  
 picDeal.CalculDeal(imgList)  
 **if** count==configIfno[**'END\_WAIT\_MAX\_TIME'**]:  
 logger.error(**'spider error closing！'**)  
 **break** time.sleep(30)  
 **if** redisCount==getKeyLen(server):  
 count = count + 1

**import** json  
**from** CommonFile **import** SAVE\_ES\_PARAMS, GetMd5  
**from** elasticsearch **import** helpers  
**from** scrapy\_redis.utils **import** bytes\_to\_str  
**from** datetime **import** datetime  
**from** elasticsearch\_dsl.connections **import** connections  
  
  
**class** DealRedis(object):  
  
 **def** \_\_init\_\_(self, server, logger, \*\*kwargs):  
 self.server = server  
 self.bulkElems = []  
 params = kwargs[**'esConnect'**]  
 self.es = connections.create\_connection(\*\*params)  
 self.esIndex = kwargs[**'INDEX'**]  
 self.esType = kwargs[**'TYPE'**]  
 self.esWeight = kwargs[**'WEIGHT'**]  
 self.reItemName = kwargs[**'reItemName'**]  
 self.esLog = logger  
  
 **def** ReadRedis(self, lens):  
 **for** i **in** range(lens):  
 data = self.server.lpop(self.reItemName)  
 **if** data:  
 self.ReadData(data)  
 **if** len(self.bulkElems):  
 self.SaveBulk()  
  
 **def** ReadData(self, data):  
 data = bytes\_to\_str(data)  
 data = json.loads(data)  
 data = self.CreateEsType(data)  
 self.bulkElems.append(data)  
  
 **def** SaveBulk(self):  
 **try**:  
 helpers.bulk(self.es, self.bulkElems)  
 **except**:  
 idList = [item[**'\_id'**] **for** item **in** self.bulkElems **if '\_id' in** item]  
 lines = json.dumps(idList, ensure\_ascii=**False**)  
 self.esLog.error(lines)  
 **finally**:  
 **del** self.bulkElems[:len(self.bulkElems)]  
  
 **def** ReadRemain(self):  
 data = self.server.lpop(self.reItemName)  
 **while** data:  
 self.ReadData(data)  
 data = self.server.lpop(self.reItemName)  
 **if** len(self.bulkElems):  
 self.SaveBulk()  
  
 **def** GenSuggests(self, index, infoTuple):  
 usedWords = set()  
 suggests = []  
 **try**:  
 **for** text, weight **in** infoTuple:  
 **if** text:  
 words = self.es.indices.analyze(index=index,  
 body={**"analyzer"**: **"ik\_max\_word"**, **'filter'**: [**"lowercase"**],  
 **'text'**: text})  
 anylyzedWords = set([r[**"token"**] **for** r **in** words[**"tokens"**] **if** len(r[**"token"**]) > 1])  
 newWords = anylyzedWords - usedWords  
 **else**:  
 newWords = set()  
  
 **if** newWords:  
 usedWords = set.union(usedWords, newWords)  
 suggests.append({**"input"**: list(newWords), **"weight"**: weight})  
 **except**:  
 self.esLog.error(**"create suggestion errror!"**)  
 **return** suggests  
  
 **def** CreateEsType(self, jsonData):  
 url = jsonData[**'url'**]  
 esId = GetMd5(url)  
 title = jsonData[**'title'**] **if 'title' in** jsonData **else ""** urlOrigin = jsonData[**'urlOrigin'**] **if 'urlOrigin' in** jsonData **else ""** isIndex=jsonData[**'isIndex'**] **if 'isIndex' in** jsonData **else False** content = jsonData[**'content'**] **if 'content' in** jsonData **else ""** imageResult = jsonData[**'imageResult'**] **if 'imageResult' in** jsonData **else ""** createStr=**"0001-1-1"  
 if 'createdDate' in** jsonData:  
 createStr=jsonData[**'createdDate'**]  
 **try**:  
 createdDate = datetime.strptime(createStr, **'%Y-%m-%d'**)  
 **except**:  
 createdDate=datetime.strptime(**"0001-1-1"**, **'%Y-%m-%d'**)  
 saveDate = datetime.today()  
 elem = {  
 **"\_index"**: self.esIndex,  
 **"\_type"**: self.esType,  
 **"\_id"**: esId,  
 **"\_source"**: {  
 **"url"**: url,  
 **"title"**: title,  
 **"url\_origin"**: urlOrigin,  
 **"content"**: content,  
 **"img\_download"**:imageResult,  
 **"is\_index"**:isIndex,  
 **"create\_date"**: createdDate,  
 **"save\_date"**: saveDate,  
 **"suggest"**: self.GenSuggests(self.esIndex, ((title, self.esWeight),))  
 }  
 }  
 **return** elem  
  
 @classmethod  
 **def** CreateDealRedis(cls, server, logger):  
 **return** cls(server, logger, \*\*SAVE\_ES\_PARAMS)

**import** numpy **as** np  
**from** numpy **import** linalg  
**from** keras.applications.vgg16 **import** VGG16  
**from** keras.preprocessing **import** image  
**from** keras.applications.vgg16 **import** preprocess\_input  
**from** CommonFile **import** VGG\_PARAMS  
**from** PIL **import** Image **as** pilImage  
  
  
**class** VGGNet:  
 **def** \_\_init\_\_(self, \*\*kwargs):  
 self.inputShape = kwargs[**'inputShape'**]  
 self.weight = kwargs[**'weight'**]  
 self.pooling = kwargs[**'pooling'**]  
 self.includeTop = kwargs[**'includeTop'**]  
 self.modelVGG = VGG16(weights=self.weight, input\_shape=self.inputShape, pooling=self.pooling, include\_top=**False**)  
  
 **def** VGGExtractFeat(self, imgPath):  
 img = image.load\_img(imgPath)  
 imgSize = img.size  
 changeSize = (self.inputShape[0], self.inputShape[1])  
 **if** imgSize != changeSize:  
 img = img.resize(changeSize, pilImage.NEAREST)  
 img = image.img\_to\_array(img)  
 img = np.expand\_dims(img, axis=0)  
 img = preprocess\_input(img)  
 feat = self.modelVGG.predict(img)  
 normFeat = feat[0] / linalg.norm(feat[0])  
 **return** normFeat, imgSize  
  
 @classmethod  
 **def** CreateVGGNet(cls):  
 **return** cls(\*\*VGG\_PARAMS)

**import** os  
  
*# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)*BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
JSON\_DATA = os.path.join(BASE\_DIR,**'static/js/JsonData'**)  
  
UPLOAD\_DIR = os.path.join(BASE\_DIR,**'static/uploadImg/'**)  
LINK\_DIR = os.path.join(BASE\_DIR,**'static/'**)  
SEARCH\_DIR = os.path.join(BASE\_DIR,**'static/Result/\*'**)  
H5PY\_FILE = os.path.join(os.path.dirname(BASE\_DIR),**'VGGGFatureCNN.h5'**)  
  
*# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/1.11/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!*SECRET\_KEY = **'5)$op9fxf2b#%(\*\_-qcr7sf)\*c@gr!v=d851(\*3f\*2gef0f!#d'***# SECURITY WARNING: don't run with debug turned on in production!*DEBUG = **False***# Application definition*INSTALLED\_APPS = [  
 **'django.contrib.admin'**,  
 **'django.contrib.auth'**,  
 **'django.contrib.contenttypes'**,  
 **'django.contrib.sessions'**,  
 **'django.contrib.messages'**,  
 **'django.contrib.staticfiles'**,  
 **'search'**,  
]  
  
MIDDLEWARE = [  
 **'django.middleware.security.SecurityMiddleware'**,  
 **'django.contrib.sessions.middleware.SessionMiddleware'**,  
 **'django.middleware.common.CommonMiddleware'**,  
 **'django.middleware.csrf.CsrfViewMiddleware'**,  
 **'django.contrib.auth.middleware.AuthenticationMiddleware'**,  
 **'django.contrib.messages.middleware.MessageMiddleware'**,  
 **'django.middleware.clickjacking.XFrameOptionsMiddleware'**,  
]  
  
ROOT\_URLCONF = **'HfutSearch.urls'**TEMPLATES = [  
 {  
 **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  
 **'DIRS'**: [os.path.join(BASE\_DIR, **'templates'**)]  
 ,  
 **'APP\_DIRS'**: **True**,  
 **'OPTIONS'**: {  
 **'context\_processors'**: [  
 **'django.template.context\_processors.debug'**,  
 **'django.template.context\_processors.request'**,  
 **'django.contrib.auth.context\_processors.auth'**,  
 **'django.contrib.messages.context\_processors.messages'**,  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = **'HfutSearch.wsgi.application'***# Database  
# https://docs.djangoproject.com/en/1.11/ref/settings/#databases  
  
  
  
# Password validation  
# https://docs.djangoproject.com/en/1.11/ref/settings/#auth-password-validators*AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.MinimumLengthValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.CommonPasswordValidator'**,  
 },  
 {  
 **'NAME'**: **'django.contrib.auth.password\_validation.NumericPasswordValidator'**,  
 },  
]  
  
  
*# Internationalization  
# https://docs.djangoproject.com/en/1.11/topics/i18n/*LANGUAGE\_CODE = **'en-us'**TIME\_ZONE = **'UTC'**USE\_I18N = **True**USE\_L10N = **True**USE\_TZ = **True**ALLOWED\_HOSTS = [**'\*'**]  
*# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/1.11/howto/static-files/*STATIC\_URL = **'/static/'**STATIC\_ROOT=os.path.join(BASE\_DIR, **"static"**)  
  
CACHES = {  
 **"default"**: {  
 **"BACKEND"**: **"django\_redis.cache.RedisCache"**,  
 **"LOCATION"**: **"redis://127.0.0.1:6379"**,  
 **"OPTIONS"**: {  
 **"CLIENT\_CLASS"**: **"django\_redis.client.DefaultClient"** }  
 }  
}

*"""***from** django.conf.urls **import** url  
**from** search.views **import** SearchSuggest, SearchView, IndexView, SearchPictureView, \  
 UploadPictureMiddle, AjaxPictureMiddle,TxtPicSearch,AjaxCollegeMiddle  
**from** django.views.static **import** serve  
**from** HfutSearch.settings **import** STATIC\_ROOT  
urlpatterns = [  
 url(**r'^$'**, IndexView.as\_view(), name=**"index"**),  
  
 url(**r'^searchPicture/(?P<uploadType>.\*)/$'**, SearchPictureView.as\_view(), name=**"searchPicture"**),  
  
 url(**r'^uploadPicture/(?P<uploadType>.\*)/$'**, UploadPictureMiddle.as\_view(), name=**"uploadPicture"**),  
  
 url(**r'^txtPicSearch/$'**, TxtPicSearch.as\_view(), name=**"txtPicSearch"**),  
  
 url(**r'^ajaxQuest/$'**, AjaxPictureMiddle.as\_view(), name=**"ajaxQuest"**),  
  
 url(**r'^ajaxCollege/$'**, AjaxCollegeMiddle.as\_view(), name=**"ajaxCollege"**),  
  
 url(**r'^suggest/$'**, SearchSuggest.as\_view(), name=**"suggest"**),  
  
 url(**r'^static/(?P<path>.\*)$'**,serve,{**'document\_root'**:STATIC\_ROOT}),  
  
 url(**r'^search/$'**, SearchView.as\_view(), name=**"search"**),  
]

**import** os  
  
**from** django.core.wsgi **import** get\_wsgi\_application  
  
os.environ.setdefault(**"DJANGO\_SETTINGS\_MODULE"**, **"HfutSearch.settings"**)  
  
application = get\_wsgi\_application()

**import** urllib  
**from** django.core.files.storage **import** default\_storage  
**from** django.core.files.base **import** ContentFile  
**from** django.conf **import** settings  
**import** h5py  
**import** glob  
**import** os  
**import** uuid  
**import** codecs  
**import** json  
**import** hashlib  
**import** re  
VGG\_PARAMS = {  
 **'weight'**: **'imagenet'**,  
 **'inputShape'**: (224, 224, 3),  
 **'includeTop'**: **False**,  
 **'pooling'**: **'max'**,  
}  
connStr={  
 **"host"**: **"localhost"**,  
 **"port"**: 16110  
 }  
FETCH\_NUMBER\_DEF = {