day09_爬虫文档解析整合&数据保存准备

目标

- 能够完成爬虫初始化url的解析代码
- 能够完成个人空间页的解析
- 能够完成文章目标页的解析
- 能够进行整合测试
- 能够编写频道的保存及查询

1 文档解析

1.1解析规则工具类ParseRuleUtils

com.heima.crawler.utils.ParseRuleUtils

```
public class ParseRuleUtils {


    /**
     * 获取有效的抓取规则
     *
     * @param html
     * @param parseRuleList
     * @return
     */
    public static List<ParseRule> parseHtmlByRuleList(Html html, List<ParseRule>
parseRuleList) {
        List<ParseRule> effectiveGrapRuleList = null;
        if (null != html && null != parseRuleList && !parseRuleList.isEmpty()) {
            //对内容的解析
            List<ParseRule> ruleList = parseContent(html, parseRuleList);
            //对数据有效性的校验
            effectiveGrapRuleList = getEffectiveParseRuleList(ruleList);
        }
        return effectiveGrapRuleList;
    }


    /**
     * 获取有效的抓取规则
     *
     * @return
     */
    private static List<ParseRule> getEffectiveParseRuleList(List<ParseRule>
parseRuleList) {
        List<ParseRule> effectiveParseRuleList = new ArrayList<ParseRule>();
        if (null != parseRuleList && !parseRuleList.isEmpty()) {
            for (ParseRule parseRule : parseRuleList) {
```

```
                if (parseRule.isAvailability()) {
                    effectiveParseRuleList.add(parseRule);
                }
            }
        }
        return effectiveParseRuleList;
    }

    /**
     * 获取有效的抓取规则
     *
     * @return
     */
    public static boolean validateUrl(List<String> urlList, List<ParseRule>
parseRuleList) {
        boolean flag = false;
        if (null != urlList && !urlList.isEmpty()) {
            for (String url : urlList) {
                for (ParseRule parseRule : parseRuleList) {
                    //获取URL校验的正则表达式
                    String validateRegular = parseRule.getUrlValidateRegular();
                    boolean validateResult = true;
                    if (StringUtils.isNotEmpty(validateRegular)) {
                        try {
                            //通过正则表达式进行校验
                            validateResult = Pattern.matches(validateRegular,
url);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }

                    if (validateResult) {
                        flag = true;
                        break;
                    }
                }
                if (flag) {
                    break;
                }
            }
        }
        return flag;
    }


    /**
     * 获取抓取的内容
     *
     * @param html
     * @param parseRuleList
     * @return
     */
    private static List<ParseRule> parseContent(Html html, List<ParseRule>
parseRuleList) {
        if (null != html && null != parseRuleList && !parseRuleList.isEmpty()) {
            for (ParseRule parseRule : parseRuleList) {
                List<String> contentList = null;
```

```
                //Css表达式的解析
                if (CrawlerEnum.ParseRuleType.CSS ==
parseRule.getParseRuleType()) {
                    contentList = html.css(parseRule.getRule()).all();
                    //正则表达式的解析
                } else if (CrawlerEnum.ParseRuleType.REGULAR ==
parseRule.getParseRuleType()) {
                    contentList = html.regex(parseRule.getRule()).all();
                    //Xpath 表达式的解析
                } else if (CrawlerEnum.ParseRuleType.XPATH ==
parseRule.getParseRuleType()) {
                    contentList = html.xpath(parseRule.getRule()).all();
                }
                if (null != contentList && !contentList.isEmpty()) {
                    parseRule.setParseContentList(contentList);
                }
            }
        }
        return parseRuleList;
    }

    /**
     * 将内容转换为链接地址
     *
     * @param contentList
     * @return
     */
    public static List<String> getUrlLinks(List<String> contentList) {
        List<String> urlList = new ArrayList<String>();
        if (null != contentList && !contentList.isEmpty()) {
            for (String content : contentList) {
                urlList.addAll(Html.create(content).links().all());
            }
        }
        return urlList;
    }

}
```

CrawlerConfig添加字段

com.heima.crawler.config.CrawlerConfig

```
private Spider spider;

public Spider getSpider() {
    return spider;
}

public void setSpider(Spider spider) {
    this.spider = spider;
}
```

AbstractProcessFlow添加方法

```
@Autowired
private CrawlerConfig crawlerConfig;

@Autowired
private CrawlerHelper crawlerHelper;

/**
     * UA
     * user agent 意思是用户代理。用户代理是一种对数据打包、创造分组头，以及编址、传递消息的部件。
     * 用户代理是指浏览器,它的信息包括硬件平台、系统软件、应用软件和用户个人偏好.用户代理，它还包括搜索引擎。
     */
private final String UserAgent[] = {
    "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_8; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50",
    "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50",
    "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729; InfoPath.3; rv:11.0) like Gecko",
    "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.87 Safari/537.36 OPR/37.0.2178.32",
    "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 UBrowser/5.6.12150.8 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2486.0 Safari/537.36 Edge/13.10586",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.87 Safari/537.36 OPR/37.0.2178.32",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142 Safari/537.36"
};

/**
     * 设置 Accept
     * <p>
     * Accept 请求头用来告知客户端可以处理的内容类型，这种内容类型用MIME类型来表示。借助内容协商机制，服务器可以从诸多备选项中选择一项进行应用，
     * 并使用 Content-Type 应答头通知客户端它的选择。浏览器会基于请求的上下文来为这个请求头设置合适的值，比如获取一个CSS层叠样式表时值与获取图片、视频或脚本文件时的值是不同的。
     */
private final String Accept[] = {

 "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3"
};


/**
```

```
     * UserAgent 参数设置
     */
public final String UserAgentParameterName = "User-Agent";


/**
     * UserAgent 参数设置
     */
public final String AcceptParameterName = "Accept";

/**
     * 获取header头
     *
     * @return
     */
public Map<String, String> getHeaderMap() {
    Map<String, String> headerMap = new HashMap<String, String>();
    headerMap.put(UserAgentParameterName, getUserAgent());
    headerMap.put(AcceptParameterName, getAccept());
    return headerMap;
}

/**
     * request 封装
     *
     * @param url
     * @return
     */
public Request getRequest(String url) {
    Map<String, String> headerMap = getHeaderMap();
    Request request = RequestUtils.requestPackage(url, headerMap);
    return request;
}

/**
     * request 封装
     *
     * @param parseItem
     * @return
     */
public Request getRequest(ParseItem parseItem) {
    Request request = null;
    String initialUrl = parseItem.getInitialUrl();
    if (StringUtils.isNotEmpty(initialUrl)) {
        request = getRequest(initialUrl);
        crawlerHelper.setParseItem(request, parseItem);
    }
    return request;
}

/**
     * 添加request
     *
     * @param parseItemList
     */
public void addSpiderRequest(List<ParseItem> parseItemList) {
    if (null != parseItemList && !parseItemList.isEmpty()) {
        for (ParseItem parseItem : parseItemList) {
```

```
            Request request = getRequest(parseItem);
            crawlerConfig.getSpider().addRequest(request);
        }
    }
}


/**
     * 获取随机UA
     */
public String getUserAgent() {
    return UserAgent[(int) (Math.random() * (UserAgent.length))];
}

/**
     * 获取随机Accept
     */
public String getAccept() {
    return Accept[(int) (Math.random() * (Accept.length))];
}
```

1.2 抽象页面处理器AbstractCrawlerPageProcessor

com.heima.crawler.process.processor.AbstractCrawlerPageProcessor

```
/**
 * 页面数据处理类，将通过 ProxyHttpClientDownloader 类下载的数据进行解析处理，
 * 从crawlerConfigProperty配置中拿到解析表达式进行页面数据的解析
 * <p>
 * help页面 和 target 页面
 * <p>
 * 对于博客页，HelpUrl是列表页，TargetUrl是文章页。
 * 对于论坛，HelpUrl是帖子列表，TargetUrl是帖子详情。
 * 对于电商网站，HelpUrl是分类列表，TargetUrl是商品详情。
 */
@Log4j2
public abstract class AbstractCrawlerPageProcessor extends AbstractProcessFlow
implements PageProcessor {


    @Autowired
    private CrawlerHelper crawlerHelper;


    public void handel(ProcessFlowData processFlowData) {

    }

    /**
     * process是定制爬虫逻辑的核心接口，在这里编写抽取逻辑
     *
     * @param page
```

```
     */
     @Override
    public void process(Page page) {

    }


    /**
     * 抽象处理类
     *
     * @param page
     */
    public abstract void handelPage(Page page);

    /**
     * 是否需要处理类型
     *
     * @return
     */
    public abstract boolean isNeedHandelType(String handelType);


    /**
     * 是否需要处理类型
     *
     * @return
     */
    public abstract boolean isNeedDocumentType(String documentType);

    /**
     * 获取 Site 信息
     *
     * @return
     */
    public Site getSite() {
        Site site =
Site.me().setRetryTimes(getRetryTimes()).setRetrySleepTime(getRetrySleepTime()).
setSleepTime(getSleepTime()).setTimeOut(getTimeOut());
        //header 配置
        Map<String, String> headerMap = getHeaderMap();
        if (null != headerMap && !headerMap.isEmpty()) {
            for (Map.Entry<String, String> entry : headerMap.entrySet()) {
                site.addHeader(entry.getKey(), entry.getValue());
            }
        }
        return site;
    }

    /**
     * 添加数据到爬取列表
     * @param urlList 需要爬取的URL列表
     * @param request 上一个爬取的对象
     * @param documentType 需要处理的文档类型
     */
    public void addSpiderRequest(List<String> urlList, Request request,
CrawlerEnum.DocumentType documentType) {
        if (null != urlList && !urlList.isEmpty()) {
            List<ParseItem> parseItemList = urlList.stream().map(url -> {
```

```
                CrawlerParseItem parseItem = new CrawlerParseItem();
                parseItem.setUrl(url);
                String handelType = crawlerHelper.getHandelType(request);
                parseItem.setDocumentType(documentType.name());
                parseItem.setHandelType(handelType);
                return parseItem;
            }).collect(Collectors.toList());
            addSpiderRequest(parseItemList);
        }
    }

    /**
     * 获取url列表
     *
     * @param helpParseRuleList
     * @return
     */
    public List<String> getHelpUrlList(List<ParseRule> helpParseRuleList) {
        List<String> helpUrlList = new ArrayList<String>();
        for (ParseRule parseRule : helpParseRuleList) {
            List<String> urlList =
ParseRuleUtils.getUrlLinks(parseRule.getParseContentList());
            helpUrlList.addAll(urlList);
        }
        return helpUrlList;
    }


    /**
     * 重试次数
     *
     * @return
     */
    public int getRetryTimes() {
        return 3;
    }

    /**
     * 重试间隔时间 ms
     *
     * @return
     */
    public int getRetrySleepTime() {
        return 1000;
    }

    /**
     * 抓取间隔时间
     *
     * @return
     */
    public int getSleepTime() {
        return 1000;
    }

    /**
     * 超时时间
     *
```

```
     * @return
     */
    public int getTimeOut() {
        return 10000;
    }

    /**
     * 该类的组件类型
     *
     * @return
     */
    public CrawlerEnum.ComponentType getComponentType() {
        return CrawlerEnum.ComponentType.PAGEPROCESSOR;
    }

}
```

1.3 初始化URL解析 CrawlerInitPageProcessor

> 从初始化URL下载的页面中个人空间的URL并加入到下载列表，并设置解析类型为帮助页

```
/**
 * 抓取初始的页面
 */
@Component
@Log4j2
public class CrawlerInitPageProcessor extends AbstractCrawlerPageProcessor {

    @Autowired
    private CrawlerConfigProperty crawlerConfigProperty;

    /**
     * 处理数据
     *
     * @param page
     */
    @Override
    public void handelPage(Page page) {
        String initXpath = crawlerConfigProperty.getInitCrawlerXpath();
        //抓取帮助页面List
        List<String> helpUrlList =
page.getHtml().xpath(initXpath).links().all();
        addSpiderRequest(helpUrlList, page.getRequest(),
CrawlerEnum.DocumentType.HELP);
    }

    /**
     * 需要处理的爬取类型
     * 初始化只处理 正向爬取
     *
     * @param handelType
     * @return
     */
```

```
    @Override
    public boolean isNeedHandelType(String handelType) {
        return CrawlerEnum.HandelType.FORWARD.name().equals(handelType);
    }

    /**
     * 需要处理的文档类型
     * 只处理初始化的URK
     *
     * @param documentType
     * @return
     */
    @Override
    public boolean isNeedDocumentType(String documentType) {
        return CrawlerEnum.DocumentType.INIT.name().equals(documentType);
    }

    /**
     * 优先级
     *
     * @return
     */
    @Override
    public int getPriority() {
        return 100;
    }
}
```

1.4 抓取帮助页面CrawlerHelpPageProcessor

(1)前置工作

配置AbstractProcessFlow

com.heima.crawler.process.AbstractProcessFlow

```
@Autowired
private CookieHelper cookieHelper;

@Autowired
private SeleniumClient seleniumClient;

@Autowired
private CrawlerProxyProvider crawlerProxyProvider;


/**
     * 获取原始的Html 页面数据
     *
     * @param url
     * @param parameterMap
     * @return
     */
public String getOriginalRequestHtmlData(String url, Map<String, String>
parameterMap) {
    //获取代理
```

```
    CrawlerProxy proxy = crawlerProxyProvider.getRandomProxy();

    //获取Cookie列表
    List<CrawlerCookie> cookieList = cookieHelper.getCookieEntity(url, proxy);
    //通过HttpClient方式来获取数据
    String htmlData = getHttpClientRequestData(url, parameterMap, cookieList,
proxy);
    boolean isValidate =
crawlerHelper.getDataValidateCallBack().validate(htmlData);
    if (!isValidate) {
        CrawlerHtml crawlerHtml = getSeleniumRequestData(url, parameterMap,
proxy);
        htmlData = crawlerHtml.getHtml();
    }
    return htmlData;
}


/**
     * 通过Http Client 来获取数据
     *
     * @param url           请求的URL
     * @param parameterMap 参数
     * @param cookieList   cookie列表
     * @param crawlerProxy 代理
     * @return
     */
public String getHttpClientRequestData(String url, Map<String, String>
parameterMap, List<CrawlerCookie> cookieList, CrawlerProxy crawlerProxy) {
    CookieStore cookieStore = getCookieStore(cookieList);
    String jsonDate = null;
    HttpHost proxy = null;
    if (null != crawlerProxy) {
        proxy = CrawlerProxyFactory.getHttpHostProxy(crawlerProxy);
    }
    try {
        long currentTime = System.currentTimeMillis();
        log.info("HttpClient 请求数据,url:{},parameter:{},cookies:{},proxy:{}",
url, parameterMap, JSON.toJSONString(cookieList), proxy);
        jsonDate = HttpClientUtils.get(url, parameterMap, getHeaderMap(),
cookieStore, proxy, "UTF-8");
        log.info("HttpClient 请求数据完成：url:{},parameter:{},cookies:{},proxy:
{},duration:{},result:{}", url, parameterMap, JSON.toJSONString(cookieList),
proxy, System.currentTimeMillis() - currentTime, jsonDate);
    } catch (IOException e) {
        log.error("HttpClient 请求数据异常,url:{},parameter:{},cookies:{},proxy:
{},errorMsg:{}", url, parameterMap, JSON.toJSONString(cookieList), proxy,
e.getMessage());
    } catch (URISyntaxException e) {
        log.error("HttpClient 请求数据异常,url:{},parameter:{},cookies:{},proxy:
{},errorMsg:{}", url, parameterMap, JSON.toJSONString(cookieList), proxy,
e.getMessage());
    }
    return jsonDate;
}


/**
```

```
     * 获取 SeleniumRequestData
     *
     * @param url
     * @param parameterMap
     * @return
     */
public CrawlerHtml getSeleniumRequestData(String url, Map<String, String>
parameterMap, CrawlerProxy proxy) {
    String buildUrl = HttpClientUtils.buildGetUrl(url, parameterMap,
HttpClientUtils.utf8);
    String cookieName = cookieHelper.getCookieName();
    CrawlerHtml crawlerHtml = seleniumClient.getCrawlerHtml(buildUrl, proxy,
cookieName);
    if (null != crawlerHtml) {
        cookieHelper.updateCookie(crawlerHtml.getCrawlerCookieList(), proxy);
    }
    return crawlerHtml;
}


/**
     * cookie 转 CookieStore
     *
     * @param cookieList
     * @return
     */
private CookieStore getCookieStore(List<CrawlerCookie> cookieList) {
    BasicCookieStore cookieStore = null;
    if (null != cookieList && !cookieList.isEmpty()) {
        for (CrawlerCookie cookie : cookieList) {
            if (null != cookie) {
                BasicClientCookie basicClientCookie = new
BasicClientCookie(cookie.getName(), cookie.getValue());
                basicClientCookie.setDomain(cookie.getDomain());
                basicClientCookie.setPath(cookie.getPath());
                cookieStore = new BasicCookieStore();
                cookieStore.addCookie(basicClientCookie);
            }
        }
    }
    return cookieStore;
}
```

(2)CrawlerHelpPageProcessor类

com.heima.crawler.process.processor.impl.CrawlerHelpPageProcessor

```
/**
 * 抓取帮助页面
 */
@Component
@Log4j2
public class CrawlerHelpPageProcessor extends AbstractCrawlerPageProcessor {

    /**
     * 帮助页面的后缀
     */
```

```
    private final String helpUrlSuffix = "?utm_source=feed";
    /**
     * 帮助页面分页后缀
     */
    private final String helpPagePagingSuffix = "/article/list/";

    @Autowired
    private CrawlerConfigProperty crawlerConfigProperty;

    @Autowired
    private CrawlerHelper crawlerHelper;

    /**
     * 处理数据
     * @param page
     */
    @Override
    public void handelPage(Page page) {
        String handelType = crawlerHelper.getHandelType(page.getRequest());
        long currentTime = System.currentTimeMillis();
        String requestUrl = page.getUrl().get();
        log.info("开始解析帮助页数据，url:{},handelType：{}", requestUrl,
handelType);

        //获取配置的抓取规则
        String helpCrawlerXpath = crawlerConfigProperty.getHelpCrawlerXpath();

        List<String> helpUrlList =
page.getHtml().xpath(helpCrawlerXpath).links().all();
        Integer crawlerHelpNextPagingSize =
crawlerConfigProperty.getCrawlerHelpNextPagingSize();
        if (null != crawlerHelpNextPagingSize && crawlerHelpNextPagingSize > 1)
{
            List<String> docPagePagingUrlList =
getDocPagePagingUrlList(requestUrl, crawlerHelpNextPagingSize);
            if (null != docPagePagingUrlList && !docPagePagingUrlList.isEmpty())
{
                helpUrlList.addAll(docPagePagingUrlList);
            }
        }
        addSpiderRequest(helpUrlList, page.getRequest(),
CrawlerEnum.DocumentType.PAGE);
        log.info("解析帮助页数据完成，url:{},handelType:{},耗时：{}", page.getUrl(),
handelType, System.currentTimeMillis() - currentTime);
    }

    /**
     * 获取分页后的数据
     * @param url 处理的URL
     * @param pageSize 分页页数
     * @return
     */
    private List<String> getDocPagePagingUrlList(String url, int pageSize) {
        List<String> docPagePagingUrlList = null;
        if (url.endsWith(helpUrlSuffix)) {
            List<String> pagePagingUrlList = generateHelpPagingUrl(url,
pageSize);
            docPagePagingUrlList = getHelpPagingDocUrl(pagePagingUrlList);
```

```
        }
        return docPagePagingUrlList;
    }

    /**
     * 生成分页URL
     * @param url 初始URL
     * @param pageSize 分页页数
     * @return
     */
    public List<String> generateHelpPagingUrl(String url, int pageSize) {
        String pageUrl = url.replace(helpUrlSuffix, helpPagePagingSuffix);
        List<String> pagePagingUrlList = new ArrayList<String>();
        for (int i = 2; i <= pageSize; i++) {
            pagePagingUrlList.add(pageUrl + i);
        }
        return pagePagingUrlList;
    }

    /**
     * 获取分页后获取的URL
     * @param pagePagingUrlList
     * @return
     */
    public List<String> getHelpPagingDocUrl(List<String> pagePagingUrlList) {
        long currentTime = System.currentTimeMillis();
        log.info("开始进行分页抓取Doc页面");
        List<String> docUrlList = new ArrayList<>();
        int fialCount = 0;
        if (!pagePagingUrlList.isEmpty()) {
            for (String url : pagePagingUrlList) {
                log.info("开始进行Help页面分页处理，url:{}", url);
                String htmlData = getOriginalRequestHtmlData(url, null);
                boolean isValidate =
crawlerHelper.getDataValidateCallBack().validate(htmlData);
                if (isValidate) {
                    List<String> urlList = new
Html(htmlData).xpath(crawlerConfigProperty.getHelpCrawlerXpath()).links().all();
                    if (!urlList.isEmpty()) {
                        docUrlList.addAll(urlList);
                    } else {
                        fialCount++;
                        if (fialCount > 2) {
                            break;
                        }
                    }

                }
            }
        }
        log.info("分抓取Doc页面完成，耗时:{}", System.currentTimeMillis() -
currentTime);
        return docUrlList;
    }

    /**
     * 处理的爬取类型
     * 只处理正向爬取
```

```
     * @param handelType
     * @return
     */
    @Override
    public boolean isNeedHandelType(String handelType) {
        return CrawlerEnum.HandelType.FORWARD.name().equals(handelType);
    }

    /**
     * 处理的文档类型
     * 只处理帮助页面
     * @param documentType
     * @return
     */
    @Override
    public boolean isNeedDocumentType(String documentType) {
        return CrawlerEnum.DocumentType.HELP.name().equals(documentType);
    }

    @Override
    public int getPriority() {
        return 110;
    }
}
```

1.5 模板文档解析CrawlerDocPageProcessor

> 将下载的数据最终解析出来并交给下一级解析处理器解析

com.heima.crawler.process.processor.impl.CrawlerDocPageProcessor

```
/**
 * 文档页面抓取
 */
@Component
@Log4j2
public class CrawlerDocPageProcessor extends AbstractCrawlerPageProcessor {

    @Autowired
    private CrawlerConfigProperty crawlerConfigProperty;

    @Autowired
    private CrawlerHelper crawlerHelper;

    /**
     * 处理页面数据
     * @param page
     */
    @Override
    public void handelPage(Page page) {
        long currentTime = System.currentTimeMillis();
        String handelType = crawlerHelper.getHandelType(page.getRequest());
        log.info("开始解析目标页数据，url:{},handelType:{}", page.getUrl(),
handelType);
        //获取抓取规则列表
        List<ParseRule> targetParseRuleList =
crawlerConfigProperty.getTargetParseRuleList();
```

```
        //抽取有效的数据
        targetParseRuleList = ParseRuleUtils.parseHtmlByRuleList(page.getHtml(),
targetParseRuleList);
        if (null != targetParseRuleList && !targetParseRuleList.isEmpty()) {
            for (ParseRule parseRule : targetParseRuleList) {
                //将数据添加进page里面，交给后续的pipline 处理
                log.info("添加数据字段到field，url:{}，handelType:{},field:{}",
page.getUrl(), handelType, parseRule.getField());
                page.putField(parseRule.getField(),
parseRule.getMergeContent());
            }
        }

        log.info("解析目标页数据完成，url:{},handelType:{},耗时：{}", page.getUrl(),
handelType, System.currentTimeMillis() - currentTime);
    }

    /**
     * 需要处理的爬取类型
     * 所以的爬取类型都处理
     * @param handelType
     * @return
     */
    @Override
    public boolean isNeedHandelType(String handelType) {
        return true;
    }

    /**
     * 需要处理的文档类型
     * 只处理目标类型
     * @param documentType
     * @return
     */
    @Override
    public boolean isNeedDocumentType(String documentType) {
        return CrawlerEnum.DocumentType.PAGE.name().equals(documentType);
    }


    @Override
    public int getPriority() {
        return 120;
    }
}
```

1.6 文档管理器CrawlerPageProcessorManager

> 文档处理管理器将三种文档处理器结合起来进行有序的处理，从初始化URL解析到帮助页以及最终URL的解析

com.heima.crawler.process.processor.CrawlerPageProcessorManager

```
/**
```

```
 * 爬虫流程管理类
 */
@Component
public class CrawlerPageProcessorManager {

    @Autowired
    private CrawlerHelper crawlerHelper;
    @Resource
    private List<AbstractCrawlerPageProcessor> abstractCrawlerPageProcessorList;

    /**
     * 初始化注入的接口排序
     */
    @PostConstruct
    private void initProcessingFlow() {
        if (null != abstractCrawlerPageProcessorList &&
!abstractCrawlerPageProcessorList.isEmpty()) {
            abstractCrawlerPageProcessorList.sort(new Comparator<ProcessFlow>()
{
                public int compare(ProcessFlow p1, ProcessFlow p2) {
                    if (p1.getPriority() > p2.getPriority()) {
                        return 1;
                    } else if (p1.getPriority() < p2.getPriority()) {
                        return -1;
                    }
                    return 0;
                }
            });
        }
    }

    /**
     * 处理数据
     *
     * @param page
     */
    public void handel(Page page) {
        String handelType = crawlerHelper.getHandelType(page.getRequest());
        String documentType = crawlerHelper.getDocumentType(page.getRequest());
        for (AbstractCrawlerPageProcessor abstractCrawlerPageProcessor :
abstractCrawlerPageProcessorList) {
            boolean isNeedHandelType =
abstractCrawlerPageProcessor.isNeedHandelType(handelType);
            boolean isNeedDocumentType =
abstractCrawlerPageProcessor.isNeedDocumentType(documentType);
            if (isNeedHandelType && isNeedDocumentType) {
                abstractCrawlerPageProcessor.handelPage(page);
            }
        }
    }
}
```

AbstractCrawlerPageProcessor类配置

com.heima.crawler.process.processor.AbstractCrawlerPageProcessor

对其中 process方法进行补全

```
@Autowired
private CrawlerPageProcessorManager crawlerPageProcessorManager;
/**
     * process是定制爬虫逻辑的核心接口，在这里编写抽取逻辑
     *
     * @param page
     */
@Override
public void process(Page page) {
    long currentTime = System.currentTimeMillis();
    String handelType = crawlerHelper.getHandelType(page.getRequest());
    log.info("开始解析数据页面，url:{},handelType:{}", page.getUrl(), handelType);
    crawlerPageProcessorManager.handel(page);
    log.info("解析数据页面完成，url:{},handelType:{},耗时：{}", page.getUrl(),
handelType, System.currentTimeMillis() - currentTime);
}
```

2 整合

2.1实体类

CrawlerComponent类

com.heima.crawler.process.entity.CrawlerComponent

```
/**
 * 抓取组件
 */
@Setter
@Getter
public class CrawlerComponent implements Serializable {
    /**
     * 页面处理类
     */
    private PageProcessor pageProcessor;
    /**
     * pipelineList 处理
     */
    private List<Pipeline> pipelineList = new ArrayList<Pipeline>();

    /**
     * 去重组件
     */
    private Scheduler scheduler;

    /**
     * 下载组件
     */
    private Downloader downloader;


    public void addPipeline(Pipeline pipeline) {
        pipelineList.add(pipeline);
```

```
    }

}
```

2.2 流程管理器ProcessingFlowManager

com.heima.crawler.manager.ProcessingFlowManager

```
/**
 * 前置数据处理
 * 对ProcessFlow 接口类型的类进行前置实例化做一些前置处理
 * 例如AbstractOriginalDataProcess 类的 handel 方式 初始化URL 以及初始化 代理数据
 * 并生成Spider 并自定启动
 * 是爬虫服务的入口
 */
@Component
@Log4j2
public class ProcessingFlowManager {

    @Autowired
    private CrawlerConfig crawlerConfig;

    /**
     * 注入实现ProcessFlow 接口的所有类
     */
    @Resource
    private List<ProcessFlow> processFlowList;

    @Autowired
    private CrawlerNewsAdditionalService crawlerNewsAdditionalService;

    /**
     * spring 启动的时候就会进行调用
     * 对实现ProcessFlow接口的类根据getPriority() 接口对实现类进行从小到大的排序
     * 实现有序的责任链模式 一个模块处理一件事然后将数据传递到下个模块交给下各模块进行处理
     */
    @PostConstruct
    private void initProcessingFlow() {
        if (null != processFlowList && !processFlowList.isEmpty()) {
            processFlowList.sort(new Comparator<ProcessFlow>() {
                public int compare(ProcessFlow p1, ProcessFlow p2) {
                    if (p1.getPriority() > p2.getPriority()) {
                        return 1;
                    } else if (p1.getPriority() < p2.getPriority()) {
                        return -1;
                    }
                    return 0;
                }
            });
        }
        Spider spider = configSpider();
        crawlerConfig.setSpider(spider);
    }
```

```
    /**
     * 抓取组件封装
     * <p>
     * 根据接口 CrawlerEnum.ComponentType getComponentType()获取的
CrawlerEnum.ComponentType 类封装组件CrawlerComponent
     *
     * @param processFlowList
     * @return
     */
    private CrawlerComponent getComponent(List<ProcessFlow> processFlowList) {
        CrawlerComponent component = new CrawlerComponent();
        for (ProcessFlow processingFlow : processFlowList) {
            if (processingFlow.getComponentType() ==
CrawlerEnum.ComponentType.PAGEPROCESSOR) {
                component.setPageProcessor((PageProcessor) processingFlow);
            } else if (processingFlow.getComponentType() ==
CrawlerEnum.ComponentType.PIPELINE) {
                component.addPipeline((Pipeline) processingFlow);
            } else if (processingFlow.getComponentType() ==
CrawlerEnum.ComponentType.SCHEDULER) {
                component.setScheduler((Scheduler) processingFlow);
            } else if (processingFlow.getComponentType() ==
CrawlerEnum.ComponentType.DOWNLOAD) {
                component.setDownloader((Downloader) processingFlow);
            }
        }
        return component;
    }


    private Spider configSpider() {
        Spider spider = initSpider();
        spider.thread(5);
        return spider;
    }

    /**
     * 根据ProcessFlow接口getComponentType() 接口类型数生成Spider
     *
     * @param
     * @return
     */
    private Spider initSpider() {
        Spider spider = null;
        CrawlerComponent component = getComponent(processFlowList);
        if (null != component) {
            PageProcessor pageProcessor = component.getPageProcessor();
            if (null != pageProcessor) {
                spider = Spider.create(pageProcessor);
            }
            if (null != spider && null != component.getScheduler()) {
                spider.setScheduler(component.getScheduler());
            }
            if (null != spider && null != component.getDownloader()) {
                spider.setDownloader(component.getDownloader());
            }
```

```
            List<Pipeline> pipelineList = component.getPipelineList();
            if (null != spider && null != pipelineList &&
!pipelineList.isEmpty()) {
                for (Pipeline pipeline : pipelineList) {
                    spider.addPipeline(pipeline);
                }
            }
        }
        return spider;
    }


    /**
     * 正向处理
     */
    public void handel() {
        startTask(null, CrawlerEnum.HandelType.FORWARD);
    }

    /**
     * 逆向处理
     */
    public void reverseHandel() {
        List<ParseItem> parseItemList =
crawlerNewsAdditionalService.queryIncrementParseItem(new Date());
        handelReverseData(parseItemList);
        log.info("开始进行数据反向更新，增量数据数量：{}", parseItemList.size());
        if (null != parseItemList && !parseItemList.isEmpty()) {
            startTask(parseItemList, CrawlerEnum.HandelType.REVERSE);
        } else {
            log.info("增量数据为空不进行增量数据更新");
        }
    }

    /**
     * 反向同步数据处理
     *
     * @param parseItemList
     */
    public void handelReverseData(List<ParseItem> parseItemList) {
        if (null != parseItemList && !parseItemList.isEmpty()) {
            for (ParseItem item : parseItemList) {
                item.setDocumentType(CrawlerEnum.DocumentType.PAGE.name());
                item.setHandelType(CrawlerEnum.HandelType.REVERSE.name());
            }
        }
    }


    /**
     * 开始处理爬虫任务
     *
     * @param parseItemList 处理初始化URL列表
     * @param handelType    FORWARD 正向处理 REVERSE 逆向处理
     */
    public void startTask(List<ParseItem> parseItemList, CrawlerEnum.HandelType
handelType) {
        ProcessFlowData processFlowData = new ProcessFlowData();
```

```
        processFlowData.setHandelType(handelType);
        processFlowData.setParseItemList(parseItemList);
        for (ProcessFlow processingFlow : processFlowList) {
            processingFlow.handel(processFlowData);
        }
        crawlerConfig.getSpider().start();
    }
}
```

2.3 测试

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class ProcessingFlowManagerTest {

    @Autowired
    private ProcessingFlowManager processingFlowManager;

    @Test
    public void test(){
        processingFlowManager.handel();
        try {
            Thread.sleep(Integer.MAX_VALUE);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
}
```

3.数据保存准备

3.1频道

3.1.1 思路分析

- 从文章解析中获取具体的标签labels，查询数据库如果没有则需要新建
- 根据文章的labels去查找对应的频道信息

3.1.2 实体类

AdLabel类

com.heima.model.admin.pojos.AdLabel

```
@Data
public class AdLabel {
    private Integer id;
    private String name;
    private Boolean type;
    private Date createdTime;
}
```

AdChannelLabel类

com.heima.model.admin.pojos.AdChannelLabel

```
@Data
public class AdChannelLabel {
    private Integer id;
    private Integer channelId;
    private Integer labelId;
    private Integer ord;
}
```

3.1.3 mapper接口定义

AdLabelMapper

com.heima.model.mappers.admin.AdLabelMapper

```
public interface AdLabelMapper {

    int deleteByPrimaryKey(Integer id);

    int insert(AdLabel record);

    int insertSelective(AdLabel record);

    AdLabel selectByPrimaryKey(Integer id);

    int updateByPrimaryKeySelective(AdLabel record);

    int updateByPrimaryKey(AdLabel record);

    List<AdLabel> queryAdLabelByLabels(List<String> labelList);

    List<AdLabel> queryAdLabelByLabelIds(List<String> labelList);
}
```

AdChannelLabelMapper接口

com.heima.model.mappers.admin.AdChannelLabelMapper

```
public interface AdChannelLabelMapper {

    int deleteByPrimaryKey(Integer id);

    int insert(AdChannelLabel record);

    int insertSelective(AdChannelLabel record);
```

```
    AdChannelLabel selectByPrimaryKey(Integer id);

    int updateByPrimaryKeySelective(AdChannelLabel record);

    int updateByPrimaryKey(AdChannelLabel record);

    /**
     * 根据labelId查询
     * @param id
     * @return
     */
    AdChannelLabel selectByLabelId(Integer id);
}
```

AdLabelMapper.xml

mappers/admin/AdLabelMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.heima.model.mappers.admin.AdLabelMapper">
    <resultMap id="BaseResultMap" type="com.heima.model.admin.pojos.AdLabel">
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="type" property="type"/>
        <result column="created_time" property="createdTime"/>
    </resultMap>
    <sql id="Base_Column_List">
    id, name, type, created_time
  </sql>
    <select id="selectByPrimaryKey" resultMap="BaseResultMap"
parameterType="java.lang.Integer">
        select
        <include refid="Base_Column_List"/>
        from ad_label
        where id = #{id}
    </select>
    <select id="queryAdLabelByLabels" resultMap="BaseResultMap"
parameterType="list">
        select
        <include refid="Base_Column_List"/>
        from ad_label
        where name IN
        <foreach item="item" index="index" collection="list" open="("
separator="," close=")">
            #{item}
        </foreach>
    </select>
    <select id="queryAdLabelByLabelIds" resultMap="BaseResultMap"
parameterType="list">
        select
        <include refid="Base_Column_List"/>
        from ad_label
        where id IN
```

```
        <foreach item="item" index="index" collection="list" open="("
separator="," close=")">
            #{item}
        </foreach>
    </select>
    <delete id="deleteByPrimaryKey" parameterType="java.lang.Integer">
    delete from ad_label
    where id = #{id}
  </delete>
    <insert id="insert" parameterType="com.heima.model.admin.pojos.AdLabel">
    insert into ad_label (id, name, type, created_time
      )
    values (#{id}, #{name}, #{type}, #{createdTime}
      )
  </insert>
    <insert id="insertSelective"
parameterType="com.heima.model.admin.pojos.AdLabel">
        insert into ad_label
        <trim prefix="(" suffix=")" suffixOverrides=",">
            <if test="id != null">
                id,
            </if>
            <if test="name != null">
                name,
            </if>
            <if test="type != null">
                type,
            </if>
            <if test="createdTime != null">
                created_time,
            </if>
        </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="id != null">
                #{id},
            </if>
            <if test="name != null">
                #{name},
            </if>
            <if test="type != null">
                #{type},
            </if>
            <if test="createdTime != null">
                #{createdTime},
            </if>
        </trim>
    </insert>
    <update id="updateByPrimaryKeySelective"
parameterType="com.heima.model.admin.pojos.AdLabel">
        update ad_label
        <set>
            <if test="name != null">
                name = #{name},
            </if>
            <if test="type != null">
                type = #{type},
            </if>
            <if test="createdTime != null">
```

```
                created_time = #{createdTime},
            </if>
        </set>
        where id = #{id}
    </update>
    <update id="updateByPrimaryKey"
parameterType="com.heima.model.admin.pojos.AdLabel">
    update ad_label
    set name = #{name},
      type = #{type},
      created_time = #{createdTime}
    where id = #{id}
  </update>
</mapper>
```

AdChannelLabelMapper.xml

mappers/admin/AdChannelLabelMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.heima.model.mappers.admin.AdChannelLabelMapper">
    <resultMap id="BaseResultMap"
type="com.heima.model.admin.pojos.AdChannelLabel">
        <id column="id" property="id"/>
        <result column="channel_id" property="channelId"/>
        <result column="label_id" property="labelId"/>
        <result column="ord" property="ord"/>
    </resultMap>
    <sql id="Base_Column_List">
    id, channel_id, label_id, ord
  </sql>
    <select id="selectByPrimaryKey" resultMap="BaseResultMap"
parameterType="java.lang.Integer">
        select
        <include refid="Base_Column_List"/>
        from ad_channel_label
        where id = #{id}
    </select>
    <select id="selectByLabelId" resultMap="BaseResultMap"
parameterType="java.lang.Integer">
        select
        <include refid="Base_Column_List"/>
        from ad_channel_label
        where label_id = #{id}
    </select>
    <delete id="deleteByPrimaryKey" parameterType="java.lang.Integer">
    delete from ad_channel_label
    where id = #{id}
  </delete>
    <insert id="insert"
parameterType="com.heima.model.admin.pojos.AdChannelLabel">
    insert into ad_channel_label (id, channel_id, label_id,
      ord)
    values (#{id}, #{channelId}, #{labelId},
      #{ord})
```

```
  </insert>
    <insert id="insertSelective"
parameterType="com.heima.model.admin.pojos.AdChannelLabel">
        insert into ad_channel_label
        <trim prefix="(" suffix=")" suffixOverrides=",">
            <if test="id != null">
                id,
            </if>
            <if test="channelId != null">
                channel_id,
            </if>
            <if test="labelId != null">
                label_id,
            </if>
            <if test="ord != null">
                ord,
            </if>
        </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="id != null">
                #{id},
            </if>
            <if test="channelId != null">
                #{channelId},
            </if>
            <if test="labelId != null">
                #{labelId},
            </if>
            <if test="ord != null">
                #{ord},
            </if>
        </trim>
    </insert>
    <update id="updateByPrimaryKeySelective"
            parameterType="com.heima.model.admin.pojos.AdChannelLabel">
        update ad_channel_label
        <set>
            <if test="channelId != null">
                channel_id = #{channelId},
            </if>
            <if test="labelId != null">
                label_id = #{labelId},
            </if>
            <if test="ord != null">
                ord = #{ord},
            </if>
        </set>
        where id = #{id}
    </update>
    <update id="updateByPrimaryKey"
parameterType="com.heima.model.admin.pojos.AdChannelLabel">
    update ad_channel_label
    set channel_id = #{channelId},
      label_id = #{labelId},
      ord = #{ord}
    where id = #{id}
  </update>
</mapper>
```

3.1.4 service代码

AdLabelService

com.heima.crawler.service.AdLabelService

```
public interface AdLabelService {
    /**
     *param:标签的列表，逗号分隔，从文章解析中获取具体的标签labels，去数据库中查询
     return : 返回的是标签的id  以逗号分隔
     */
    public String getLableIds(String labels);
    /**
     *param:   标签id
     *return : 频道id  如果没有查到，为0  未分类
     */
    public Integer getAdChannelByLabelIds(String labels);
}
```

AdLabelServiceImpl

com.heima.crawler.service.impl.AdLabelServiceImpl

```
@Log4j2
@Service
public class AdLabelServiceImpl implements AdLabelService {

    @Autowired
    private AdLabelMapper adLabelMapper;

    @Autowired
    private AdChannelLabelMapper adChannelLabelMapper;

    @Override
    public String getLableIds(String labels) {

        long currentTime = System.currentTimeMillis();
        log.info("获取channel信息，标签：labels：{}", labels);
        List<AdLabel> adLabelList = new ArrayList<AdLabel>();
        if (StringUtils.isNotEmpty(labels)) {
            //转换成小写
            labels = labels.toLowerCase();
            List<String> labelList = Arrays.asList(labels.split(","));
            log.info("查询label数组：{}", labelList);
            List<AdLabel> tmpLabels =
adLabelMapper.queryAdLabelByLabels(labelList);
            if (null != tmpLabels && !tmpLabels.isEmpty()) {
                adLabelList = addLabelList(tmpLabels, labelList);
            } else {
                adLabelList = addLabelList(labelList);
            }
        }
```

```
        List<String> labelIdList = adLabelList.stream().map(label ->
HMStringUtils.toString(label.getId())).collect(Collectors.toList());
        String labelIds = HMStringUtils.listToStr(labelIdList, ",");
        log.info("获取channel信息完成，标签：labels：{},labelIds:{},耗时：{}",
labels, labelIds, System.currentTimeMillis() - currentTime);
        return labelIds;
    }

    @Override
    public Integer getAdChannelByLabelIds(String labelIds) {
        Integer channelId = 0;
        try {
            channelId = getSecurityAdChannelByLabelIds(labelIds);
        } catch (Exception e) {
            log.error("获取channel信息失败，errorMsg:{}", e.getMessage());
        }
        return channelId;
    }


    private Integer getSecurityAdChannelByLabelIds(String labelIds) {
        long currentTime = System.currentTimeMillis();
        log.info("获取channel信息，标签IDS：labelIds：{}", labelIds);
        Integer channelId = 0;
        if (StringUtils.isNotEmpty(labelIds)) {
            //转换成小写
            List<String> labelList = Arrays.asList(labelIds.split(","));
            log.info("查询label数组：{}", labelList);
            List<AdLabel> adLabelList =
adLabelMapper.queryAdLabelByLabelIds(labelList);
            if (null != adLabelList && !adLabelList.isEmpty()) {
                channelId = geAdChannelIdByLabelId(adLabelList.get(0).getId());
            }
        }
        channelId = channelId == null ? 0 : channelId;

        log.info("获取channel信息完成，标签：labelIds：{},channelId:{},耗时：{}",
labelIds, channelId, System.currentTimeMillis() - currentTime);
        return channelId;
    }


    public Integer geAdChannelIdByLabelId(Integer labelId) {
        Integer channelId = 0;
        AdChannelLabel adChannelLabel =
adChannelLabelMapper.selectByLabelId(labelId);
        if (null != adChannelLabel) {
            channelId = adChannelLabel.getChannelId();
        }
        return channelId;
    }

    public List<AdLabel> addLabelList(List<AdLabel> adLabelList, List<String>
orginLabelList) {
        List<String> unAddLabelList = adLabelList.stream().map(x ->
x.getName()).filter(x ->
!orginLabelList.contains(x)).collect(Collectors.toList());
        return addLabelList(unAddLabelList);
```

```
    }

    public List<AdLabel> addLabelList(List<String> labelList) {
        List<AdLabel> adLabelList = new ArrayList<AdLabel>();
        if (null != labelList && !labelList.isEmpty()) {
            for (String label : labelList) {
                adLabelList.add(addLabel(label));
            }
        }
        return adLabelList;
    }

    /**
     * 添加label
     *
     * @param label
     */
    public AdLabel addLabel(String label) {
        AdLabel adLabel = new AdLabel();
        adLabel.setName(label);
        adLabel.setType(true);
        adLabel.setCreatedTime(new Date());
        adLabelMapper.insertSelective(adLabel);
        return adLabel;
    }
}
```

3.1.5 测试

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class AdLabelServiceTest {

    @Autowired
    private AdLabelService labelService;

    @Test
    public void testGetLabelIds(){
        String labelIds = labelService.getLabelIds("java,web,xxxxxxx");
        System.out.println(labelIds);
    }

    @Test
    public void testGetAdChannelByLabelIds(){
        Integer adChannelByLabelIds =
labelService.getAdChannelByLabelIds("1,2");
        System.out.println(adChannelByLabelIds);
    }
}
```

3.2 ip代理池

3.2.1 思路分析

3.2.2 实体类

ClIpPool类

com.heima.model.crawler.pojos.ClIpPool

```
@Data
public class ClIpPool {

    private Integer id;

    private String supplier;

    private String ip;
    /**
     * 端口号
     */
    private int port;

    /**
     * 用户名
     */
    private String username;

    /**
     * 密码
     */
    private String password;

    /**
     * 错误码
     */
    private Integer code;

    /**
     * 耗时
     */
    private Integer duration;

    /**
     * 错误信息
     */
    private String error;

    private Boolean isEnable;


    private String ranges;


    private Date createdTime;

    public Integer getId() {
        return id;
    }
```

```
    public void setId(Integer id) {
        this.id = id;
    }

    public String getSupplier() {
        return supplier;
    }

    public void setSupplier(String supplier) {
        this.supplier = supplier;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

    public int getPort() {
        return port;
    }

    public void setPort(int port) {
        this.port = port;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Boolean getEnable() {
        return isEnable;
    }

    public void setEnable(Boolean enable) {
        isEnable = enable;
    }

    public String getRanges() {
        return ranges;
    }

    public void setRanges(String ranges) {
```

```
        this.ranges = ranges;
    }

    public Date getCreatedTime() {
        return createdTime;
    }

    public void setCreatedTime(Date createdTime) {
        this.createdTime = createdTime;
    }


    public Integer getCode() {
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public Integer getDuration() {
        return duration;
    }

    public void setDuration(Integer duration) {
        this.duration = duration;
    }

    public String getError() {
        return error;
    }

    public void setError(String error) {
        this.error = error;
    }
}
```

ClIpPoolMapper

com.heima.model.mappers.crawerls.ClIpPoolMapper

```
public interface ClIpPoolMapper {

    int deleteByPrimaryKey(Integer id);

    int insert(ClIpPool record);

    int insertSelective(ClIpPool record);

    ClIpPool selectByPrimaryKey(Integer id);

    int updateByPrimaryKeySelective(ClIpPool record);

    int updateByPrimaryKey(ClIpPool record);

    /**
     * 查询所有数据
```

```
     *
     * @param record
     * @return
     */
    List<ClIpPool> selectList(ClIpPool record);

    /**
     * 查询可用的列表
     *
     * @param record
     * @return
     */
    List<ClIpPool> selectAvailableList(ClIpPool record);
}
```

3.2.3 mapper接口定义

ClIpPoolMapper.xml

mappers/crawerls/ClIpPoolMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.heima.model.mappers.crawerls.ClIpPoolMapper">
    <resultMap id="BaseResultMap" type="com.heima.model.crawler.pojos.ClIpPool">
        <id column="id" property="id"/>
        <result column="supplier" property="supplier"/>
        <result column="ip" property="ip"/>
        <result column="port" property="port"/>
        <result column="username" property="username"/>
        <result column="password" property="password"/>
        <result column="code" property="code"/>
        <result column="duration" property="duration"/>
        <result column="error" property="error"/>
        <result column="is_enable" property="isEnable"/>
        <result column="ranges" property="ranges"/>
        <result column="created_time" property="createdTime"/>
    </resultMap>

    <sql id="Base_Column_where">
        <where>
            <if test="supplier!=null and supplier!=''">
                and supplier = #{supplier}
            </if>
            <if test="ip!=null and ip!=''">
                and ip = #{ip}
            </if>
            <if test="port!=null and port!=''">
                and port = #{port}
            </if>
            <if test="username!=null and username!=''">
                and username = #{username}
            </if>
            <if test="password!=null and password!=''">
                and password = #{password}
            </if>
```

```
            <if test="code!=null and code!=''">
                and code = #{code}
            </if>
            <if test="duration!=null and duration!=''">
                and duration = #{duration}
            </if>
            <if test="error!=null and error!=''">
                and error = #{error}
            </if>
            <if test="isEnable!=null and isEnable!=''">
                and is_enable = #{isEnable}
            </if>
            <if test="ranges!=null and ranges!=''">
                and ranges = #{ranges}
            </if>
        </where>
    </sql>
    <sql id="Base_Column_List">
    id, supplier, ip, port,username,password,code,duration,error,is_enable,
ranges, created_time
  </sql>
    <select id="selectList" resultMap="BaseResultMap">
        select
        <include refid="Base_Column_List"/>
        from cl_ip_pool
        <include refid="Base_Column_where"/>
    </select>
    <select id="selectAvailableList" resultMap="BaseResultMap">
        select
        <include refid="Base_Column_List"/>
        from cl_ip_pool
        <where>
            and is_enable = true
            <if test="duration!=null and duration!=''">
                <![CDATA[
                and duration <= #{duration}
                 ]]>
            </if>
        </where>
        order by duration
    </select>
    <select id="selectByPrimaryKey" resultMap="BaseResultMap"
parameterType="java.lang.Integer">
        select
        <include refid="Base_Column_List"/>
        from cl_ip_pool
        where id = #{id}
    </select>
    <delete id="deleteByPrimaryKey" parameterType="java.lang.Integer">
    delete from cl_ip_pool
    where id = #{id}
  </delete>
    <insert id="insert" parameterType="com.heima.model.crawler.pojos.ClIpPool"
            useGeneratedKeys="true" keyProperty="id">
    insert into cl_ip_pool (id, supplier, ip,
port,username,password,code,duration,error,
      is_enable, ranges, created_time
      )
```

```
    values (#{id}, #{supplier}, #{ip}, #{port}, #{username}, #{password},#
{code},#{duration},#{error},
      #{isEnable}, #{ranges}, #{createdTime}
      )
  </insert>
    <insert id="insertSelective"
parameterType="com.heima.model.crawler.pojos.ClIpPool" keyProperty="id"
useGeneratedKeys="true">

        insert into cl_ip_pool
        <trim prefix="(" suffix=")" suffixOverrides=",">
            <if test="id != null">
                id,
            </if>
            <if test="supplier != null">
                supplier,
            </if>
            <if test="ip != null">
                ip,
            </if>
            <if test="port != null">
                port,
            </if>
            <if test="username != null">
                username,
            </if>
            <if test="password != null">
                password,
            </if>
            <if test="code != null">
                code,
            </if>
            <if test="duration != null">
                duration,
            </if>
            <if test="error != null">
                error,
            </if>
            <if test="isEnable != null">
                is_enable,
            </if>
            <if test="ranges != null">
                ranges,
            </if>
            <if test="createdTime != null">
                created_time,
            </if>
        </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="id != null">
                #{id},
            </if>
            <if test="supplier != null">
                #{supplier},
            </if>
            <if test="ip != null">
                #{ip},
            </if>
```

```
            <if test="port != null">
                #{port},
            </if>
            <if test="username != null">
                #{username},
            </if>
            <if test="password != null">
                #{password},
            </if>
            <if test="code != null">
                #{code},
            </if>
            <if test="duration != null">
                #{duration},
            </if>
            <if test="error != null">
                #{error},
            </if>
            <if test="isEnable != null">
                #{isEnable},
            </if>
            <if test="ranges != null">
                #{ranges},
            </if>
            <if test="createdTime != null">
                #{createdTime},
            </if>
        </trim>
    </insert>
    <update id="updateByPrimaryKeySelective"
parameterType="com.heima.model.crawler.pojos.ClIpPool">

        update cl_ip_pool
        <set>
            <if test="supplier != null">
                supplier = #{supplier},
            </if>
            <if test="ip != null">
                ip = #{ip},
            </if>
            <if test="port != null">
                port = #{port},
            </if>
            <if test="username != null">
                username = #{username},
            </if>
            <if test="password != null">
                password = #{password},
            </if>
            <if test="code != null">
                code = #{code},
            </if>
            <if test="duration != null">
                duration = #{duration},
            </if>
            <if test="error != null">
                error = #{error},
            </if>
```

```
            <if test="isEnable != null">
                is_enable = #{isEnable},
            </if>
            <if test="ranges != null">
                ranges = #{ranges},
            </if>
            <if test="createdTime != null">
                created_time = #{createdTime},
            </if>
        </set>
        where id = #{id}
    </update>
    <update id="updateByPrimaryKey"
parameterType="com.heima.model.crawler.pojos.ClIpPool">

    update cl_ip_pool
    set supplier = #{supplier},
      ip = #{ip},
      port = #{port},
      username = #{username},
      password = #{password},
      code = #{code},
      duration = #{duration},
      error = #{error},
      is_enable = #{isEnable},
      ranges = #{ranges},
      created_time = #{createdTime}
    where id = #{id}
  </update>
</mapper>
```

3.2.4 service代码

CrawlerIpPoolService

com.heima.crawler.service.CrawlerIpPoolService

```
public interface CrawlerIpPoolService {

    /**
     * 保存方法
     *
     * @param clIpPool
     */
    public void saveCrawlerIpPool(ClIpPool clIpPool);

    /**
     * 检查代理Ip 是否存在
     *
     * @param host
     * @param port
     * @return
     */
    public boolean checkExist(String host, int port);

    /**
     * 更新方法
```

```
     *
     * @param clIpPool
     */
    public void updateCrawlerIpPool(ClIpPool clIpPool);

    /**
     * 查询所有数据
     *
     * @param clIpPool
     */
    public List<ClIpPool> queryList(ClIpPool clIpPool);

    /**
     * 获取可用的列表
     *
     * @return
     */
    public List<ClIpPool> queryAvailableList(ClIpPool clIpPool);


    public void delete(ClIpPool clIpPool);


    void unvailableProxy(CrawlerProxy proxy, String errorMsg);
}
```

CrawlerIpPoolServiceImpl

com.heima.crawler.service.impl.CrawlerIpPoolServiceImpl

```
@Service
public class CrawlerIpPoolServiceImpl implements CrawlerIpPoolService {
    @Autowired
    private ClIpPoolMapper clIpPoolMapper;

    @Override
    public void saveCrawlerIpPool(ClIpPool clIpPool) {
        clIpPoolMapper.insertSelective(clIpPool);
    }

    @Override
    public boolean checkExist(String host, int port) {
        ClIpPool clIpPool = new ClIpPool();
        clIpPool.setIp(host);
        clIpPool.setPort(port);
        List<ClIpPool> clIpPoolList = clIpPoolMapper.selectList(clIpPool);
        if (null != clIpPoolList && !clIpPoolList.isEmpty()) {
            return true;
        }
        return false;
    }

    @Override
    public void updateCrawlerIpPool(ClIpPool clIpPool) {
        clIpPoolMapper.updateByPrimaryKey(clIpPool);
    }
```

```
    @Override
    public List<ClIpPool> queryList(ClIpPool clIpPool) {
        return clIpPoolMapper.selectList(clIpPool);
    }

    @Override
    public List<ClIpPool> queryAvailableList(ClIpPool clIpPool) {
        return clIpPoolMapper.selectAvailableList(clIpPool);
    }

    @Override
    public void delete(ClIpPool clIpPool) {
        clIpPoolMapper.deleteByPrimaryKey(clIpPool.getId());
    }

    @Override
    public void unvailableProxy(CrawlerProxy proxy, String errorMsg) {
        ClIpPool clIpPoolQuery = new ClIpPool();
        clIpPoolQuery.setIp(proxy.getHost());
        clIpPoolQuery.setPort(proxy.getPort());
        clIpPoolQuery.setEnable(true);
        List<ClIpPool> clIpPoolList = clIpPoolMapper.selectList(clIpPoolQuery);
        if (null != clIpPoolList && !clIpPoolList.isEmpty()) {
            for (ClIpPool clIpPool : clIpPoolList) {
                clIpPool.setEnable(false);
                clIpPool.setError(errorMsg);
                clIpPoolMapper.updateByPrimaryKey(clIpPool);
            }
        }
    }
}
```

3.2.5 测试

```
```