链滴

Java 动手写爬虫: 四、日志埋点输出 & amp; 动态配置支持

作者：YiHui

原文链接：https://ld246.com/article/1501130065529

来源网站：链滴

第四篇, 日志埋点输出 & 动态配置支持

前面基本上实现了一个非常简陋的爬虫框架模型，很多关键链路都没有日志，在分析问题时，就比较烦了，因此就有了这一篇博文

其次就是解决前几篇遗留的容易解决的问题

实际上，日志的输出应该贯穿在实际的开发过程中的，由于之前写得比较随意，直接System.out了，以现在就来填坑了

1.日志埋点设计

采用 logback 左右日志输出， 这里有一篇博文可供参考 《Logback 简明使用手册》

埋点的关键链路

1. 当前爬取的任务信息
2. 爬取任务的耗时
3. 应用的状态（如爬取了多少个，还剩下多少个待爬取等）
4. 爬取结果输出
5. 其他一些信息

实现比较简单，在pom中添加依赖

```
<!--日志-->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.21</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.1.7</version>
</dependency>
```

添加配置文件

logback-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%-4relative [%thread] %-5level %logger{35} - %msg %n</pattern>
        </encoder>
    </appender>

    <logger name="com.quick.hui.crawler" level="DEBUG"/>
```

```
    <root level="INFO">
        <appender-ref ref="STDOUT"/>
    </root>
</configuration>
```

代码中埋点

.... （直接参考源码即可）

2. 爬取频率控制

很多网站会对访问的频率进行限制，这是一个最基础的防爬手段了，所以我们的爬取需要一个可以设爬取任务的频率控制

1. 设计

目的

- 采用一个比较简单的方案，每次从队列中获取爬取任务时，sleep指定的时间，来实现爬取频率的制
- 对此我们设计得稍微高级一点，这个sleep时间，我们希望是可以动态配置的

方案

采用配置项来解决这个，（为了后续的拓展，读取配置搞成面向接口的编程方式），我们先提供一个础的，根据本地配置文件来读取频率控制参数

实现

因为采用配置文件的方式，所以一个用于读取配置文件的辅助工具类是必须的

1. 配置文件读取 FileConfRead

```
@Slf4j
public class FileConfRead implements IConfRead {


    public Config initConf(String path) {
        try {
            Properties properties = read(path);

            Config config = new Config();
            config.setSleep(properties.getProperty("sleep"), 0);
            config.setEmptyQueueWaitTime(properties.getProperty("emptyQueueWaitTime"), 200)


            return config;
        } catch (Exception e) {
            log.error("init config from file: {} error! e: {}", path, e);
            return new Config();
        }
```

```
    }


    private Properties read(String fileName) throws IOException {
        try (InputStream inputStream = FileReadUtil.getStreamByFileName(fileName)) {
            Properties pro = new Properties();
            pro.load(inputStream);
            return pro;
        }
    }


    private File file;
    private long lastTime;

    public void registerCheckTask(final String path) {
        try {
            file = FileReadUtil.getFile(path);
            lastTime = file.lastModified();


            ScheduledExecutorService scheduledExecutorService = Executors.newScheduledThrea
Pool(1);
            scheduledExecutorService.scheduleAtFixedRate(() -> {
                        if (file.lastModified() > lastTime) {
                            lastTime = file.lastModified();
                            ConfigWrapper.getInstance().post(new ConfigWrapper.UpdateConfEvent());
                        }
                    },
                    1,
                    1,
                    TimeUnit.MINUTES);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

实现类主要继承接口 IConfRead, 接口中定义了两个方法，一个用于获取配置信息，一个用于注册配信息的变动监听事件

```
public interface IConfRead {

    /**
     * 初始化配置信息
     *
     * @param var
     * @return
     */
    Config initConf(String var);


    /**
```

```
    * 注册配置信息更新检测任务
    *
    * @param path
    */
    void registerCheckTask(final String path);
}
```

回到具体的实现，读取配置文件信息比较简单，直接使用jdk的Properties文件的读写方式，接下来则注册监听事件的实现上，我们的设计思路如下:

- 获取配置文件的更新时间
- 每隔一段时间主动去查看下配置文件的更新时间，判断是否更新过
    - 若更新，则重新加载配置文件，覆盖之前的
    - 若无更新，直接放过

2. 配置类 Config

这里定义所有的配置信息，方便后续的维护和查阅

```
@Getter
@Setter
@ToString
public class Config {

    /**
     * 爬取任务的间隔时间
     */
    private long sleep;


    /**
     * 从队列中获取任务，返回空时，等待时间之后再进行重试
     */
    private long emptyQueueWaitTime;


    public void setSleep(String str, long sleep) {
        this.sleep = NumUtils.str2long(str, sleep);
    }

    public void setEmptyQueueWaitTime(String str, long emptyQueueWaitTime) {
        this.emptyQueueWaitTime = NumUtils.str2long(str, emptyQueueWaitTime);
    }
}
```

3. 配置信息获取封装类 ConfigWrapper

这里封装了获取配置信息的接口，内部维护配置信息的变更事件，我们采用EventBus来实现事件的监听

```
@Slf4j
public class ConfigWrapper {
    private static final String CONFIG_PATH = "conf/crawler.properties";

    private EventBus eventBus;


    private IConfRead confRead;

    private Config config;

    private static volatile ConfigWrapper instance;

    private ConfigWrapper() {
        confRead = new FileConfRead();
        confRead.registerCheckTask(CONFIG_PATH);
        config = confRead.initConf(CONFIG_PATH);


        // 注册监听器
        eventBus = new EventBus();
        eventBus.register(this);
    }


    public static ConfigWrapper getInstance() {
        if (instance == null) {
            synchronized (ConfigWrapper.class) {
                if (instance == null) {
                    instance = new ConfigWrapper();
                }
            }
        }

        return instance;
    }


    @Subscribe
    public void init(UpdateConfEvent event) {
        config = confRead.initConf(event.conf);

        if (log.isDebugEnabled()) {
            log.debug("time:{} processor:{} update config! new config is: {}",
                    event.now, event.operator, config);
        }
    }


    public Config getConfig() {
        return config;
    }
```

```
    public void post(Object event) {
        eventBus.post(event);
    }

    @Getter
    @Setter
    public static class UpdateConfEvent {
        private long now = System.currentTimeMillis();

        private String operator = "System";

        private String conf = CONFIG_PATH;
    }
}
```

3. 源码地址

项目地址： https://github.com/liuyueyi/quick-crawler

日志埋点对应的tag: v0.006

动态配置对应的tag: v0.007

相关链接

- Java 动手写爬虫: 一、实现一个最简单爬虫
- Java 动手写爬虫: 二、 深度爬取
- Java 动手写爬虫: 三、爬取队列