

新增调整概述

主要针对新增的工作流和现场环境进行调整，以下满足现状的情况，并开发对应测试类，主要有以下调整来满足现状：

- ☒ 内网Maven缺包问题
- ☒ 新增获取工作流任务实例列表接口
- ☒ 新增获取工作流任务实例日志列表接口
- ☒ 新增获取工作流任务实例日志详情接口
- ☒ 新增根据工作流和任务名获取某一个数据时间运行日志详情
- ☒ 单独分离测试代码
- ☐ 需求是会逐渐增长的！

调整内容

调整内容主要包含以下内容：

- 测试全部分离到test下
- DEMO类独立作为API借口实现
- Maven支持外网环境打包把依赖包全部打包到lib目录下
- 内网环境可注释到Maven依赖，通过导入本地包方式导入lib目录下依赖包

新增内容

新增内容主要包含以下内容：

- DEMO类新增查看任务实例运行情况taskInstanceList方法
- DEMO类新增获取任务ID方法getTaskID
- DEMO类新增任务实例日志列表方法
- DEMO串联通过工作流ID和任务ID获取某一具体任务实例的日志信息
- 测试新增获取任务实例列表测试方法taskInstanceTest
- 测试新增获取任务实例日志详情测试方法taskInstanceLog

版本信息

当前版本为1.0.1

历史版本为1.0.0

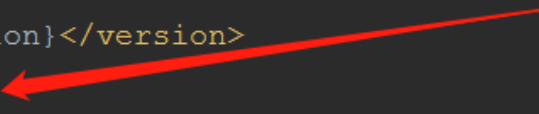
调整详情

测试代码调整

之前代码测试代码和DEMO业务实现代码内，现在测试代码必须放到test下。

1. junit依赖scope设置为test，如下

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
```



2. 新增测试类WorkFlowTest，并且迁移测试代码块到该类下

```
public class WorkFlowTest {

    /**
     * 创建工作流test
     */
    @Test
    public void createTest() { WorkflowDemo.create(properties.getProperty("tbds_workflow_create_file")); }

    /**
     * 启动工作流test
     */
    @Test
    public void startTest() {
        String workflowName = "ftp-hdfs-demo";
        String workflowId = list(workflowName);

        WorkflowDemo.start(workflowId);
    }

    /**
     * 停止工作流test
     */
    @Test
    public void stopTest() {
        String workflowName = "ftp-hdfs-demo";
        String workflowId = list(workflowName);

        WorkflowDemo.stop(workflowId);
    }
}
```

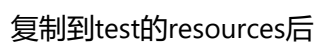
迁移的测试方法包含：

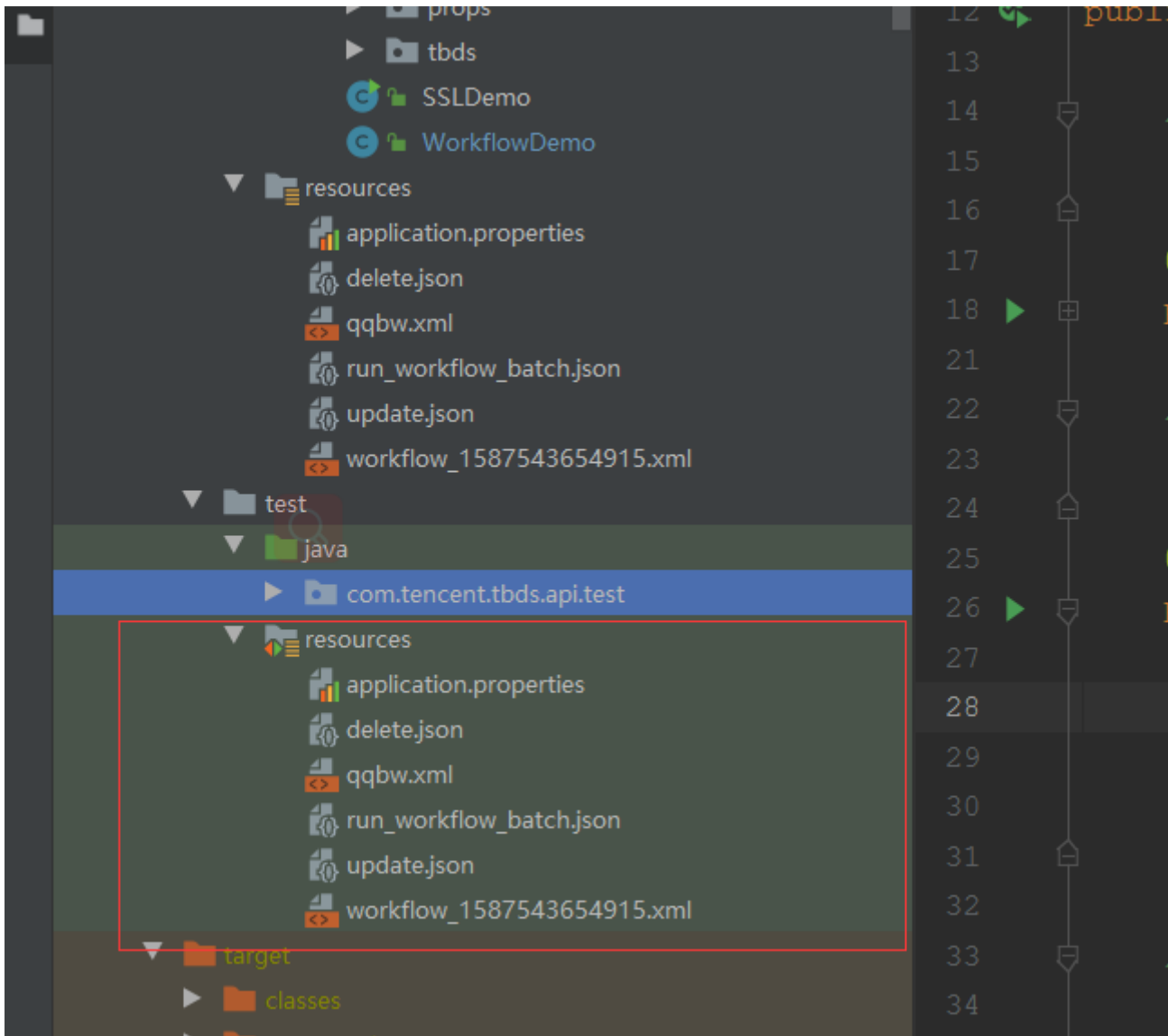
- 创建工作流 createTest
- 启动工作流 startTest
- 停止工作流 stopTest
- 删除工作流 deleteTest
- 工作流串联 workflowSeries
- 重跑工作流任务实例 triggerTaskTest

测试resources调整

之前版本直接main进行测试，那么配置main的resources即可，但是现在测试全部调整到test下，那么需要复制一份main下resource到测试的resources下。

main下resources





maven打包调整

中行现场都是内网开发，内网私库还不完善，每次折腾这个包消耗大量时间。基于这个问题maven打包进行了调整可以在外面把全部的依赖包下载到工程的lib下，这样内网即可直接注释掉maven依赖部分，直接使用lib下依赖包即可。

要实现这功能需要调整的部分有：

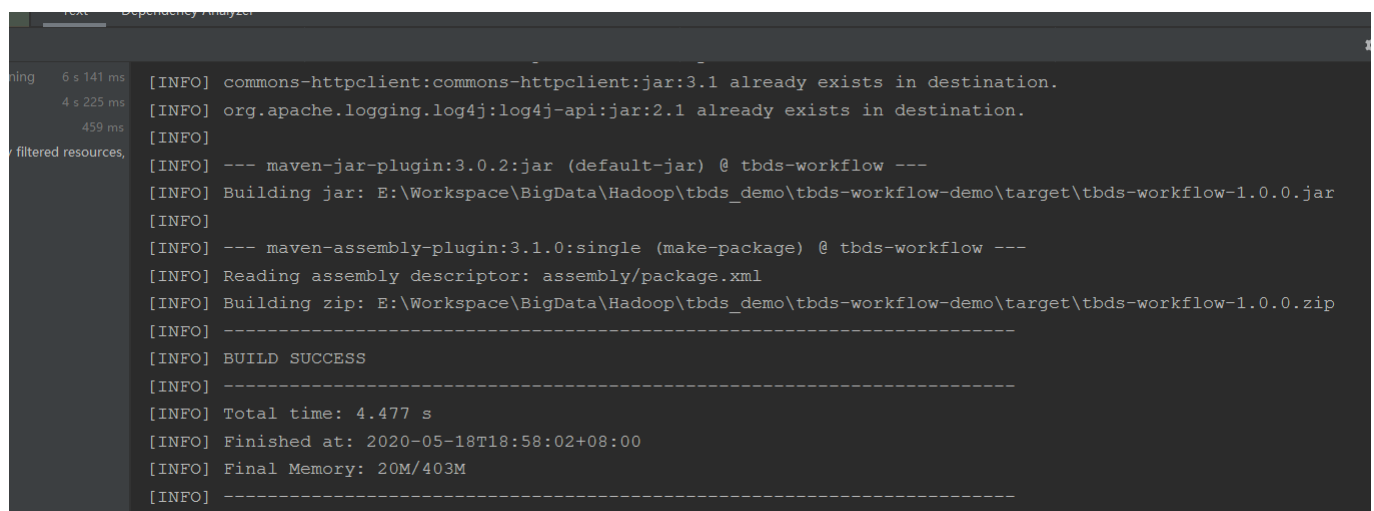
1. maven新增打包依赖包下载到lib目录下

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>3.0.0</version>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>prepare-package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.basedir}/lib</outputDirectory>
        <excludeTransitive>false</excludeTransitive>
        <stripVersion>false</stripVersion>
      </configuration>
    </execution>
  </executions>
</plugin>

```

2.外网环境进行编译打包(可不执行)，因提供包的时候已经打好依赖包到lib下



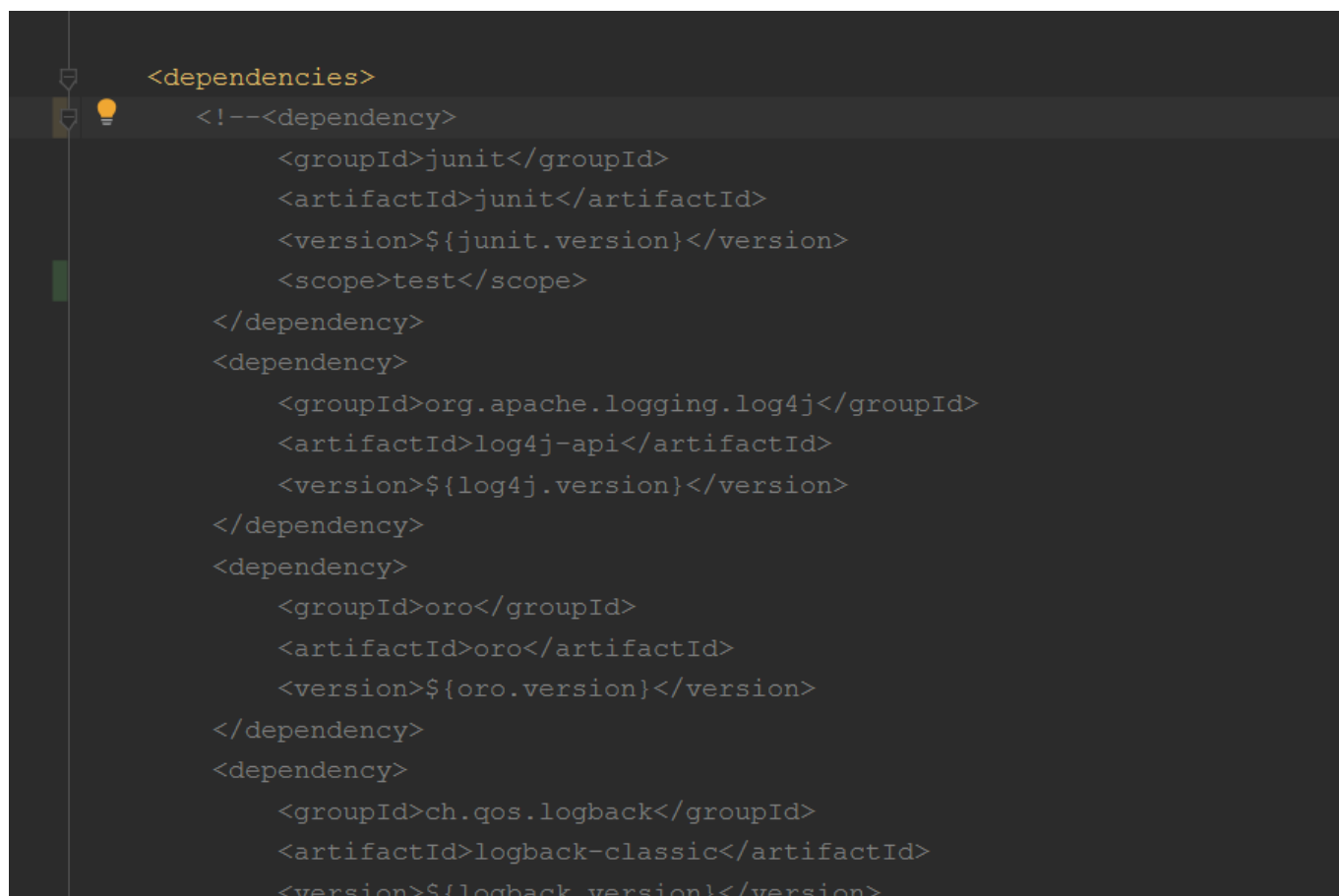
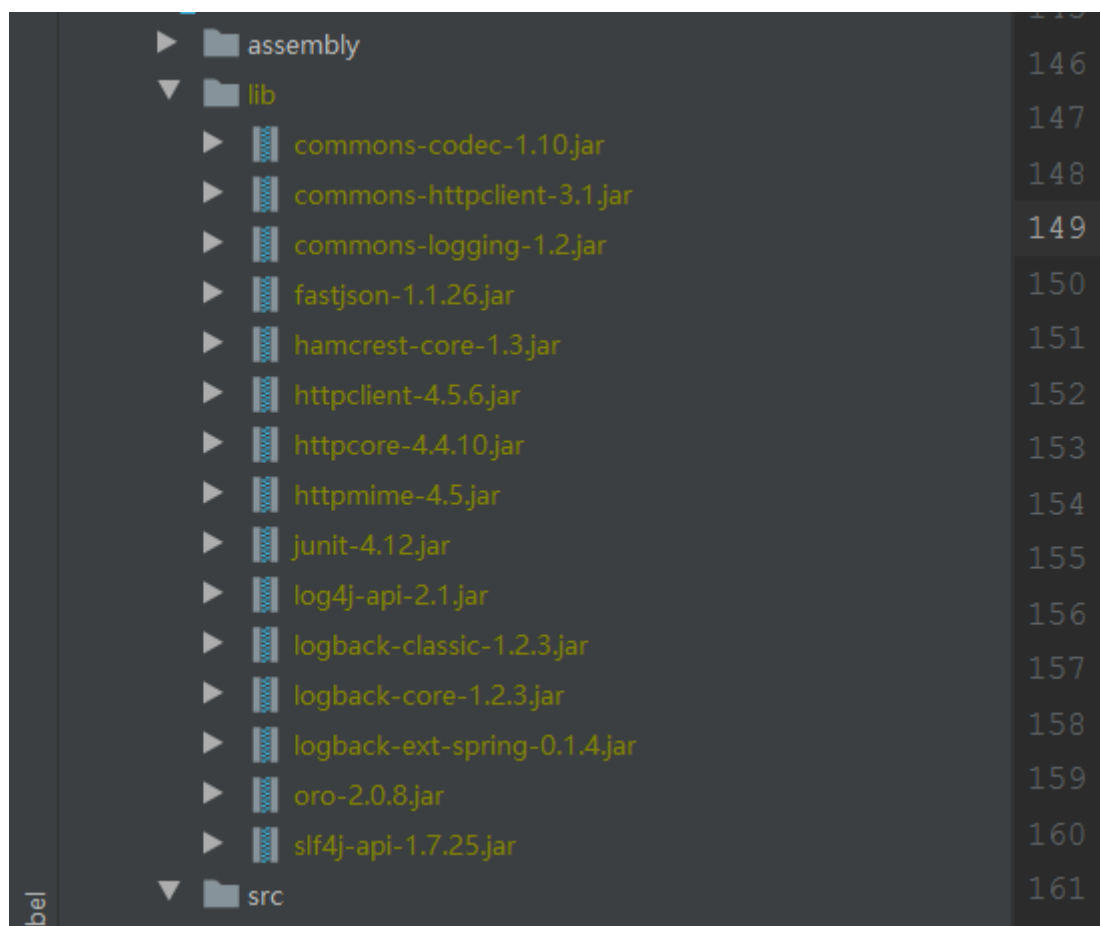
```

[INFO] commons-httpclient:commons-httpclient:jar:3.1 already exists in destination.
[INFO] org.apache.logging.log4j:log4j-api:jar:2.1 already exists in destination.
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ tbds-workflow ---
[INFO] Building jar: E:\Workspace\BigData\Hadoop\tbds_demo\tbds-workflow-demo\target\tbds-workflow-1.0.0.jar
[INFO] --- maven-assembly-plugin:3.1.0:single (make-package) @ tbds-workflow ---
[INFO] Reading assembly descriptor: assembly/package.xml
[INFO] Building zip: E:\Workspace\BigData\Hadoop\tbds_demo\tbds-workflow-demo\target\tbds-workflow-1.0.0.zip
[INFO] BUILD SUCCESS
[INFO] Total time: 4.477 s
[INFO] Finished at: 2020-05-18T18:58:02+08:00
[INFO] Final Memory: 20M/403M

```

3.内网环境注释掉maven全部依赖配置

4.Project Structure 的 Libraries添加Java配置lib目录，并且选择Model



5.查看包并检查是否已经可编译

```

<dependencies>
  <!--<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>${log4j.version}</version>
  </dependency>
  <dependency>
    <groupId>oro</groupId>
    <artifactId>oro</artifactId>
    <version>${oro.version}</version>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
  </dependency>
</dependencies>

```

新增详情

新增的内容主要是基于上传版本没有，现在比较确定的需求接口，比如任务实例列表、任务实例日志列表、任务实例日志详情等。

相关新增工作流API接口说明可参考[工作流API任务实例和日志接口说明.doc](#)相关文档。

获取任务ID方法

获取任务ID方法的方法名为getTaskID，实现功能是通过工作流名和任务名称获取对应的任务ID。

方法说明

所属类	方法名	参数说明	返回值
WorkflowDemo	getTaskID	参数有两个工作流名和任务名	任务ID

参数说明

参数名	参数类型	参数说明
workflowName	String	工作流名称
taskName	String	任务名称

功能实现：

1. 通过工作流名调用获取任务列表方法，解析结果报文获取到任务列表数组。
2. 循环任务列表数组获取到任务名是否跟输入值任务名是否相同。

3. 匹配上的解析报文中获取到任务ID。

代码实现：

```
/**
 * 获取taskID
 *
 * @param workflowName 工作流名称
 * @param taskName task 名称
 * @return 请求报文json
 */
public static String getTaskID(String workflowName, String taskName) throws
Exception {
    String taskList = queryTaskList(workflowName);
    String taskId = null;

    JSONObject jsonObject = JSONObject.parseObject(taskList);
    JSONObject resultData = jsonObject.getJSONObject("resultData");
    JSONArray content = resultData.getJSONArray("content");

    for (int i = 0; i< content.size(); i++){
        JSONObject jsonObject1 = content.getJSONObject(i);
        String taskName1 = jsonObject1.getString("taskName");
        if(taskName.equals(taskName1)){
            taskId = jsonObject1.getString("taskId");
        }
    }
    return taskId;
}
```

任务实例列表方法和测试方法

任务实例列表方法通过工作流名和任务名获取的全部的任务实例列表情况。

任务实例列表方法

方法说明

所属类	方法名	参数说明	返回值
WorkflowDemo	taskInstanceList	参数有两个工作流名和任务名	结果报文

参数说明

参数名	参数类型	参数说明
workflowName	String	工作流名称
taskName	String	任务名称

功能实现：

1. 先通过获取任务ID方法获取任务ID。
2. 通过任务ID获取和其他请求参数查询获取任务实例列表API接口。
3. 返回结果任务实例列表结果报文。

代码实现：

```
/**
 * 查看任务实例情况
 *
 * @param workflowName 工作流名
 * @param taskName 任务名称
 */
public static String taskInstanceList(String workflowName, String taskName)
throws Exception {
    String taskID = getTaskID(workflowName, taskName);
    String method = "GET";
    Map<String, String> params = new HashMap<>();
    params.put("pageIndex", "0");
    params.put("pageSize", "5");
    params.put("condition[taskParam]", taskID);
    //params.put("condition[status]", "2");
    String url = properties.getProperty("tbds_proter_ip") +
properties.getProperty("tbds_task_instance_url");
    return publicHttps(url, method, params);
}
```

特别注意：接口的URL地址、请求参数、结果报文说明请查阅**工作流API任务实例和日志接口说明.doc**文档。

任务实例列表测试方法

是进行任务列表方法的测试验证的测试方法，测试方法中工作流名和任务名称请根据实际情况调整。

测试代码：

```
@Test
public void taskInstanceTest() throws Exception {
    String workflowName = "doc";
    String taskName = "compress-doc";
    WorkflowDemo.taskInstanceList(workflowName, taskName);
}
```

特别注意：测试方法后续都在test目录下，并且测试类统一都一样为WorkFlowTest，后续不作说明。

任务实例日志列表方法和测试方法

任务实例日志列表方法

任务实例下可能存在多个日志情况，可能是人工重跑或异常重跑，如正常情况是一个任务实例是只有一个日志情况。

如需要获取某个任务实例的日志详情首先得需要获取对应的任务实例的对应的日志列表，按需求获取某个或全部的日志详情信息。 **功能实现**

1. 通过工作流名和任务名获取任务ID
2. 根据任务ID获取对应的任务实例列表
3. 如获取第一个任务实例的获取数据时间
4. 根据1中taskID和3的数据时间获取到对应任务实例全部的日志类别

代码实现

```
/**
 *
 * @param workflowName 工作流名
 * @param taskName 任务名
 *
 * @return 任务实例日志列表结果报文
 */
public static String taskInstanceLogList(String workflowName, String taskName)
throws Exception {
    String method = "GET";
    Map<String, String> params = new HashMap<>();
    params.put("taskId", getTaskID(workflowName, taskName));
    String listLogListURL = properties.getProperty("tbds_proter_ip") +
properties.getProperty("tbds_task_instance_log_list");
    String taskInstanceList = taskInstanceList(workflowName, taskName);

    JSONObject rRoot = JSONObject.parseObject(taskInstanceList);
    //任务实例JSONArray
    JSONArray content =
rRoot.getJSONObject("resultData").getJSONArray("content");

    //最新的那个任务实例getJSONObject(1)
    String dateTime = content.getJSONObject(1).get("dateTime").toString();
    params.put("dateTime", dateTime);
    return publicHttps(listLogListURL, method, params);
}
```

任务实例日志列表测试方法

测试任务实例日志类别，其中的参数工作流名和任务名称根据实际情况调整。

测试代码：

```
@Test
public void taskInstanceLogList() throws Exception {
    String workflowName = "doc";
```

```
String taskName = "compress-doc";
WorkflowDemo.taskInstanceLogList(workflowName, taskName);
}
```

任务实例日志详情方法和测试方法

需要获取根据工作流名和任务名获取任务实例的某个或多个日志详情。

任务实例日志详情方法

功能实现

1. 根据任务实例日志列表方法获取到任务实例
2. 通过任务ID和数据时间获取任务实例日志列表
3. 根据日志列表获取某个日志列表的IP和尝试次数值
4. 根据任务ID、数据时间、IP、尝试次数值获取详细日志

代码实现

```
String taskID = getTaskID(workflowName, taskName);
String method = "GET";

Map<String, String> params = new HashMap<>();
params.put("taskId", taskID);
String listLogListURL = properties.getProperty("tbds_proter_ip") +
properties.getProperty("tbds_task_instance_log_list");
String taskInstanceList = taskInstanceList(workflowName, taskName);

JSONObject rRoot = JSONObject.parseObject(taskInstanceList);
//任务实例JSONArray
JSONArray content =
rRoot.getJSONObject("resultData").getJSONArray("content");

////////////////////////////////////
////////////////////////////////////
//最新的那个任务实例getJSONObject(1)
String dataTime = content.getJSONObject(1).get("dataTime").toString();
params.put("dataTime", dataTime);
String listLogList = publicHttps(listLogListURL, method, params);
//{
//  "resultCode":"0",
//  "message":null,
//  "resultData":
//  [
//    "18K,
//    2020-05-13 09:01:22,
//    172.27.0.124,
//    1"
//  ],
//  "markInfo":null
```

```
//}  
// -----  
  
//params.put("condition[status]", "2");  
String url = properties.getProperty("tbds_proter_ip") +  
properties.getProperty("tbds_task_instance_log_info");  
//查询单个任务实例下的任务列表  
String resultData =  
JSONObject.parseObject(listLogList).getJSONArray("resultData").get(0).toString();  
  
//任务调度器的代理服务器IP，也就是任务实例执行的客户端IP  
params.put("ip", resultData.split(",")[2]);  
params.put("tries", resultData.split(",")[3]);  
  
String resultResponse = publicHttps(url, method, params);  
System.out.println(resultResponse);  
  
return resultResponse;
```

任务实例日志详情测试方法

测试方法主要测试获取到的某个任务实例的某个日志的日志详情。

测试代码

```
@Test  
public void taskInstanceLog() throws Exception {  
    String workflowName = "doc";  
    String taskName = "compress-doc";  
    WorkflowDemo.taskInstanceLogInfo(workflowName, taskName);  
}
```