作业与批处理开发框架

简介

> **简介**
>
> 采用作业调度框架开发。
>
> 总体步骤为，代码中开发Job类，然后再作业调度系统中定义此job，并定义job的schedule计划，由作业调度系统触发执行。
>
> 所有作业都按照读，算，写入的逻辑进行拆分，使用批处理框架设计。
>
> 作业处理中的进度，异常等信息，都通过作业监控页面展示。

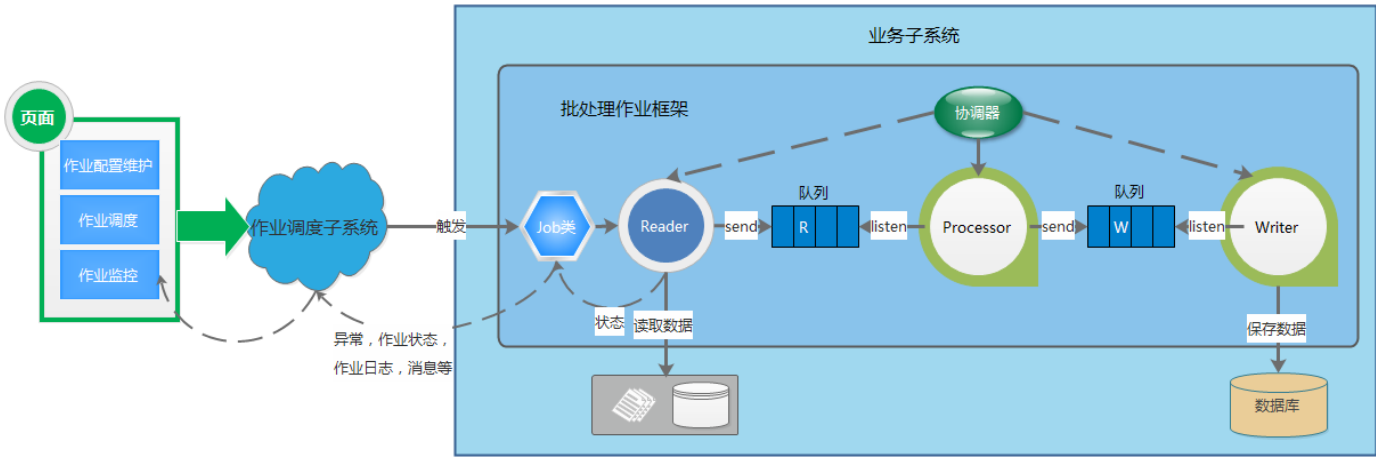
> demo代码在 http://git.acca.com.cn:7990/projects/OPRA-GIT/repos/araf-demo/browse

Maven依赖

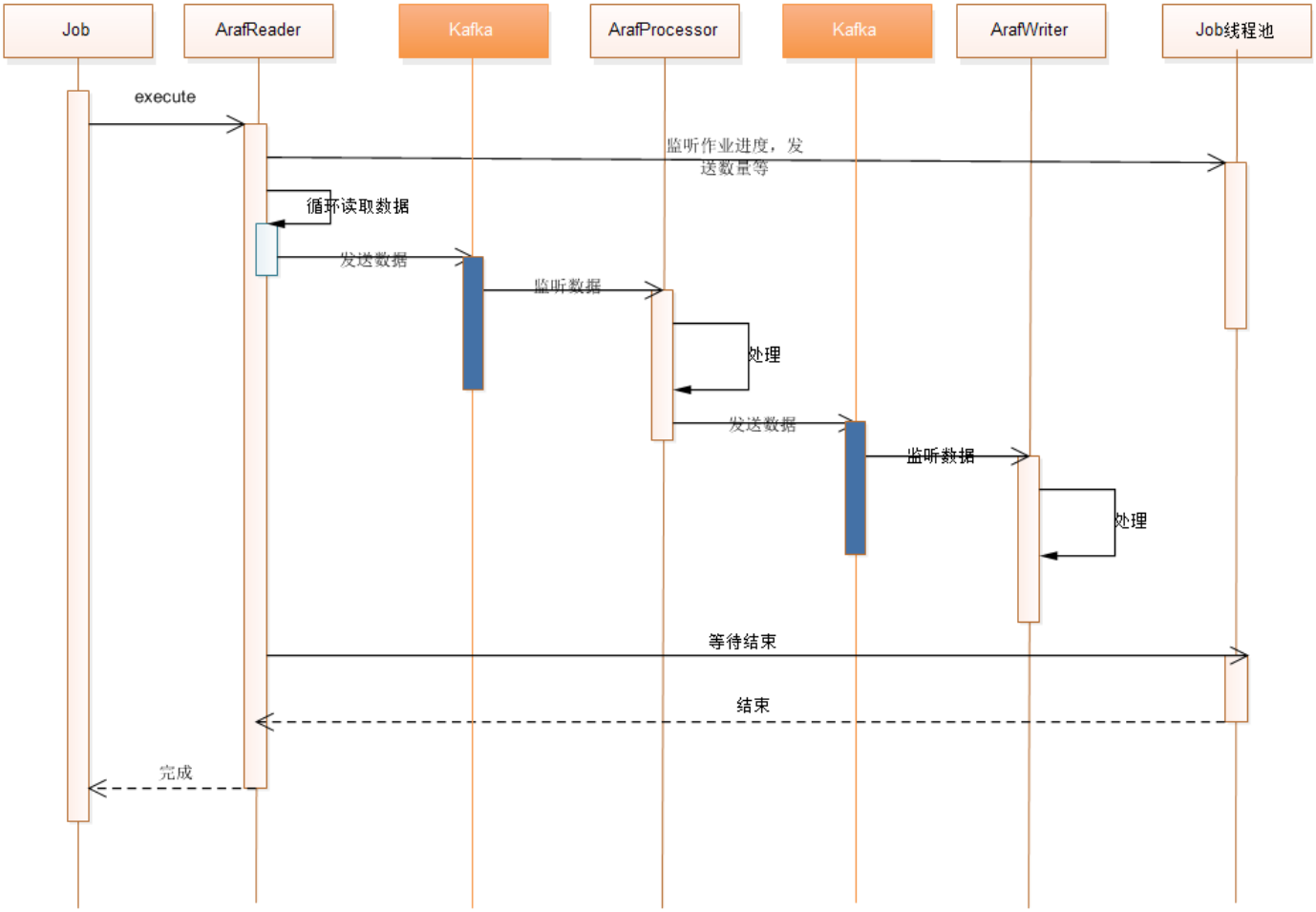
```
<dependency>
    <groupId>com.acca</groupId>
    <artifactId>araf-job-spring-boot-starter</artifactId>
</dependency>
```

逻辑架构

架构图



时序图



说明

1、Reader为单线程，reader方法需要返回需要处理的总量。Job框架自动监听处理进度，展示在页面上。

2、所有类型的processor共享processor线程池。所有类型的writer共享writer线程池。监听kafka的队列，收到消息后，采用一个内存队列将消息发送给线程池处理。

3、判断结束的逻辑：

reader发送的数量 == processor接收的数量

processor发送的数量 == writer接收的数量

规定时间内，数量对不上，则超时退出

4、所有发送的方法，都有失败的回调接口，一般情况下，业务代码无需关注并复写这些接口。

5、由于我们的业务要求准确性高，所以出现异常则作业失败，重新执行。

重构说明

1、发送消息的格式为二进制格式，比json格式，序列化发序列化性能提升。

2、增加了内存队列，控制并发的数量，防止OOM问题。

3、改进了作业完成的判断逻辑。

4、采用redis位图记录处理的数量，同时验证消息是否重复。

类描述

status类

当前作业SSIM_INPUT_JOB,/D:/Enterprise/GIT/OPRA/opra-mas-global/opra-mas-global-server/target/test-classes/dataset/inputfile/ssim/SSIM_202006-CA-SAMPLES.txt处理状态为：


```
{"PLANRENOUNCED":"0",
"READER_START":"1",
"READER_FAIL":"0",
"PROCESSOR_SEND_BIT":"2",
"ALL_END":"0",
"WRITER_PERSIST_BIT":"2",
"TIMEOUT":"0",
"WRITER_FAIL":"0",
"WRITER_RECEIVED_BIT":"2",
"PROCESSOR_ALL":"1",
"PROCESSOR_BIT":"2",
"WRITER_ALL":"1",
"READER_END":"2",
"READER_FIN":"1",
"RENOUNCED":"0",
"SIGN_DUPLICATION":"0",
"PROCESSOR_FAIL":"0",
"OTHER":"0"}
```

JobContext类

1、jobContext.setTotalCount Reader类会自动调用这个方法，用来监控作业进度。

如果不是使用Reader的Job，可以自己手工调用：

jobContext.setCompleteCount 当前成功的数量

jobContext.setTotalCount 总数量

作业框架会自动上报进度。

2、jobContext.getOutputParams() 存储需要向外传递的参数。会自动加入到后续依赖执行的作业中。

3、jobContext.getSharedVariable() 存储共享变量，注意，不能序列化的共享变量。

4、JobContext.getCurrentJobContext ， getJobContextById(jobInstanceId) 静态方法获取JobContext数据。在业务代码执行的任意位置都可以获取。默认不用传jobInstanceId.

Reader类

ArafReader<V> VStringkafka

onFail() 发送失败的事件，需要做相应的补偿操作。

```
     *  processor  writer
    * super.onFail();
    * super.onFail();
```

exec() 返回值必须与发送成功的数量一致，不一致的情况下，job会不能结束

execute() readerprocessorwriter

boolean true false isEmptyreader<= 0 isRenounce() isTimeOut

getMsg()

针对不同类型的数据包含以下几个抽象类AbstractFileReader、AbstractExcelReader、AbstractPdfReader、AbstractXmlReader、AbstractJsonReader、AbstractGeneralPartitionReader

```
@Slf4j
public abstract class ArafReader<V> implements IJobType {
...
    /**
     * Kafka.
     * @param k  sign + block .
     * @param v value.
     * @throws ExecutionException
     * @throws InterruptedException
     */
    protected SendResult<String, byte[]> send(AdpKey k, V v){
...
    }
    /**
     *  processor  writer
     * super.onFail();
     * super.onFail();
     *
     * @param k adpKey
     * @param v value
     * @param ex
     */
    protected void onFail(AdpKey k, V v, Throwable ex) {
...
    }
    /**
     * .
     *
     * @param ac
     * @return exec  <= 0  sendEnd
     */
    public abstract long exec(String sign);
    /**
     * .
     *
     * @param sign
     * @return true , false  .
     * @throws ExecutionException
     * @throws InterruptedException
     */
    public void execute(String sign) {...}
    /**
     *
     * @param sign
     * @return
     */
    public String[] getMsg(String sign) {
        return arafStatus().getMessage(getJobType(), sign);
    }
}
```

Processor类

process() null

onSendFailure() 数据发送writer失败，默认操作是输出日志及对writer数量加一操作，防止任务不能退出。

super.onSendFailure()

```
@Slf4j
public abstract class ArafProcessor<R, W> implements IJobType {

    /**
     * .
     *
     * @param k AdpKey
     * @param v
     * @return W
     */
    public abstract W process(AdpKey k, R v);

        /**
        * writerwriter
    * super.onSendFailure();
        */
    protected void onSendFailure(AdpKey k, W v, Throwable ex){}
}
```

ArafWriter类

persist

```
@Slf4j
public abstract class ArafWriter<W> implements IJobType{
   /**
     * .
     *
     * @param k AdpKey
     * @param v V
     */
    protected abstract void persist(AdpKey k, W v);
}
```

ArafBatchWriter类

persist

```
@Slf4j
public abstract class ArafWriter<W> implements IJobType{
   /**
     * .
     *
     * @param k AdpKey
     * @param v V
     */
    protected abstract void persist(List<AdpKey> keys, List<W> values);
}
```

批处理框架参数调优

参数名	默认值	说明
spring.adp.executor.processor.count	8	processor线程池大小
spring.adp.executor.writer.count	8	writer线程池大小
spring.adp.executor.batchWriter.count	8	batchWriter线程池大小
spring.adp.executor. processorBufferQueue.size	100	processor接收消息缓冲区大小。当内存足够时，调整缓冲区大小可以加快接收消息速度。
spring.adp.executor.writerBufferQueue. size	100	writer接收消息缓冲区大小。当内存足够时，调整缓冲区大小可以加快接收消息速度。
spring.adp.executor. batchWriterBufferQueue.size	100	batchWriter接收消息缓冲区大小。当内存足够时，调整缓冲区大小可以加快接收消息速度。
spring.adp.executor.batchWriter. bufferTime	200	毫秒。批量处理时，凑够批量数据的等待时间。
spring.adp.executor.batchWriter.batch. size	500	batchWriter中一次批量处理最大的消息数量。
spring.adp.timeout	300	秒。超时时间。作业框架中出现reader发送消息与processor接收；processor发送与writer接收；数量不一致时，报错的超时时间。一般都是processor处理过慢，或者writer过慢导致。
spring.kafka.max.poll.records	1000	从kafka一次获取消息的最大数量
spring.adp.partitions	map	动态调整队列的分区个数。 例如： spring: adp: partitions: 'OPRA-PARALLEL-SSIM_INPUT_JOB-read': '6' 队列准确名称可以在kafkaEagle中查询。 支持配置中心的动态刷新

作业注意事项

1、使用作业框架中，出现了业务无法处理的异常或者业务代码主动抛出的异常，则作业会快速失败。（Reader不再发送消息。）

2、调用arafStatus.renounce，则将快速放弃本次作业，作业状态为Failed。一般在业务代码中不需要自己调用，只需要抛出异常即可。

3、对于保存数据库的操作，推荐使用ArafBatchWriter，可以批量处理数据。

普通job开发

实现job接口

```
@Component(Constants.JOB_DB_DEMO_IMPORT)
@Slf4j
public class DbJob extends AbstractJob {

    @Override
    public void run(JobContext context) {
        // service

    }
}
```

文件导入类型的作业开发

设计Job类

对于文件导入的处理，需要开发一个Job子类，继承AbstractFileImportJob。代码如下：

```
@Component(Constants.JOB_OAG_IMPORT)
@Slf4j
public class OagJob extends AbstractFileImportJob {

    @Autowired
    private OagReader oagReader;

    @Autowired
    private ImportFileService importFileService;

    @Override
    public void run(JobContext context) {
        List<String> filePath = this.getFilePath(context);
        filePath.forEach(f -> {
            log.info("begin process file.");
            // sign   ,
            oagReader.execute(f);
            log.info("end process file.");
        });
    }

    @Override
    public void rollback(JobContext context) {
        List<String> filePath = this.getFilePath(context);
        filePath.forEach(f -> {
            importFileService.clearPartition(f);
        });
    }

    @Override
    public ArafReader getReader() {
        return oagReader;
    }

}
```

- @Component中定义当前作业的名称，此作业名称为String类型，需要在前台的作业调度模块维护（维护前台页面的作业配置，参见附录）。
- 实现run方法。run方法中，可以处理业务逻辑，进行文件级别的验证，如果处理中出现异常，则可以直接抛出BusinessException，作业将记录Failed状态，并通过前台页面可以查询并显示作业的日志。
- 实现rollback方法。对于每一个作业来说，管理员可以通过前台页面点击回滚操作，清理这次作业的处理痕迹。回退作业的逻辑处理，需要在rollback方法中实现。
- rerun方法，父类中有rerun方法，并且默认是先调用rollback，再调用run，如需改变，子类中复写。
- 导入文件的job类中可以调用，getFilePath(context)方法，返回值为List<String>，包含本次要导入的文件路径的列表。此数据一般由文件管理系统中的自动作业传入，或者当用户上传文件后，自动触发并传入。
- 注入一个Reader类，并调用reader类的执行方法。

实现Reader类

Reader类是用来读取文件的类。对于一般的文本文件，框架提供了AbstractFileReader，子类需要继承，并实现processLines方法。

reader子类不需要定义JobType 父类会自动从job中获取jobType。

```
```

```
@Component
@Slf4j
public class OagReader extends AbstractFileReader<SsimFlightDTO> {

    @Autowired
    private MasOagSsimVersionService masOagSsimVersionService;

    @Override
    protected long processLines(String filePath, BufferedReader reader)
throws IOException {
        String fileName = FileUtils.getFileName(filePath);
        String line = "";
        SsimFlightDTO dto = null; //
        String flightNumberAndItinerary = ""; // , 3flightDTO.
        SsimRecord2 seasonRecord = null; // season record
        SsimLegDTO legRecord = null; // record3record4
        long count = 0;
        long sendnum = 0; //
        while ((line = reader.readLine()) != null) {
            count++; //

            String firstChar = ArafStringUtils.substring(line, 0, 1);

            if ("3".equals(firstChar)) {
                String currentFlight = ArafStringUtils.substring(line, 2,
11);
                if (dto != null) {
                    if (currentFlight.equals(flightNumberAndItinerary)) {
                        // legflightDTO.
                        legRecord = new SsimLegDTO();
                        legRecord.getLines().add(new SsimLine(count,
line));
                        dto.getLegDTOs().add(legRecord);
                        continue; //
                    } else {
                        // scheduleflightDTO
                        //
                        //
                        sendnum ++;
                        this.send(AdpKey.of(filePath, sendnum), dto);
                        dto = null;
                    }

                }

                //
                flightNumberAndItinerary = currentFlight;
                dto = new SsimFlightDTO();
                dto.setRecord2(seasonRecord);
                legRecord = new SsimLegDTO();
                legRecord.getLines().add(new SsimLine(count, line));
                dto.getLegDTOs().add(legRecord);
```

```
            } else if ("4".equals(firstChar) && legRecord != null) {
                legRecord.getLines().add(new SsimLine(count, line));
            } else if ("5".equals(firstChar)) {
                //
                sendnum ++;
                this.send(AdpKey.of(filePath, sendnum), dto);
                dto = null;
            } else if ("1".equals(firstChar)) {
            } else if ("2".equals(firstChar)) {
                seasonRecord = SegmentUtils.formatBySegment(SsimRecord2.
class, line);
                seasonRecord.setFileName(fileName);
                //
                seasonRecord.setPartition(Integer.valueOf(FileUtils.
regStrNum(fileName)));
                masOagSsimVersionService.processType2(seasonRecord);
            } else {
                // 0
            }

        }

        return sendnum;
    }

}
```

- Reader是单线程读文件的。读取文件后，可以在方法中循环处理每一行数据，并进行一些必要的汇总检查等逻辑。
- Reader需要泛型一个对象，一般是DTO类，用来封装数据对象，并传输。某些文件是一行对应一个数据实体，有的则是多行对应数据实体。
- 当一个数据实体被生成，需要调用send方法发送消息，其中key值需要传入当前数据实体的顺序号，value则为封装后的数据实体。
- 文件行处理中如果需要中断，则直接抛BusinessException即可。
- reader中的execute方法执行中抛出的异常，会被框架捕获，并显示在作业监控页面中。

实现Processor类

Processor类为业务逻辑的处理类。

```
@Component
public class OagProcessor extends ArafProcessor<SsimFlightDTO, SsimDTO> {

    @Autowired
    private OagSsimService oagSsimService;

    @Override
    public SsimDTO process(AdpKey k, SsimFlightDTO v) {
        return this.processData(v);
    }

    private SsimDTO processData(SsimFlightDTO t) {
        return oagSsimService.processSsim(t);
    }

    @Override
    public String getJobType() {
        return Constants.JOB_OAG_IMPORT;
    }
}
```

- 需要实现getJobType方法，返回jobType，同Job的Component中的名字。
- 需要泛型两个DTO类型，一个是接受的Reader发送的DTO，一个是处理完之后，封装的数据库实体的DTO。
- 实现process方法。
- process中的验证等异常，则直接抛出BusinessException异常即可，框架会将异常展示到作业监控的页面中。

实现Writer类

writer为写出持久化存储的类。

```
@Component
public class OagWirter extends ArafWriter<SsimDTO> {

    @Autowired
    private MasOagSsimType3Service masOagSsimType3Service;

    @Override
    public void persist(AdpKey k, SsimDTO v) {
        this.persist(Arrays.asList(v));
    }

    public int persist(List<SsimDTO> t) {
        return masOagSsimType3Service.saveOagSsimEntity(t);
    }

    @Override
    public String getJobType() {
        return Constants.JOB_OAG_IMPORT;
    }

}
```

- 需要实现getJobType方法，返回jobType，同Job的Component中的名字。
- 需要泛型一个DTO类型，同processor发送的DTO类型
- 实现persist方法，其中不应该再包含业务逻辑，应该是调用service进行保存。注意：业务逻辑应该在processor中处理完成。

在页面配置job类

基于数据库数据的批处理作业

设计Job类

对于后台批处理的作业，需要先设计一个job类。

```
@Component(Constants.JOB_DB_DEMO_IMPORT)
@Slf4j
public class DbJob extends AbstractJob {
    @Autowired
    private DbReader dbReader;
    @Autowired
    private BlockStatus blockStatus;
    @Override
    public void run(JobContext context) {
        String sign = Uuids.shortUuid();
        log.info("begin process file.");
        // sign   ,
        dbReader.execute(sign, null);
        boolean r = blockStatus.isRenounce(dbReader.getJobType(), sign);
        if (r) {
            dbReader.cleardb(sign);
        }

    }
}
```

- @Component中标记作业名称
- 注入一个Reader

实现Reader类

继承AbstractBlockReader类，实现getTotalCount，getBatchSize的方法。

```
@Component
@Slf4j
public class DbReader extends AbstractBlockReader {

    @Override
    public Long getTotalCount(JobContext jobContext) {
        return 100l;
    }

    @Override
    public int getBatchSize(JobContext jobContext) {
        return 10;
    }

    public void cleardb(String sign) {
        // TODO Auto-generated method stub

    }

    @Override
    public String getJobType() {
        return Constants.JOB_DB_DEMO_IMPORT;
    }

}
```

- 实现getTotalCount的方法，计算本次批处理中需要计算的总数。方法中可以通过JobContext获取页面配置的作业参数。
- 实现getBatchSize，设置分片数据量的大小。

实现Processor类

继承AbstractBlockProcessor类，泛型处理后的对象类型。

```
@Component
@Slf4j
public class DbProcessor extends AbstractBlockProcessor<ArafdemoVO> {

    @Override
    public String getJobType() {
        return Constants.JOB_DB_DEMO_IMPORT;
    }

    @Override
    public List<ArafdemoVO> process(AdpKey k, CommonBlock v) {
        List<ArafdemoVO> list = new ArrayList<>();
        for (long i = v.getBegin(); i < v.getBatchSize(); i++) {
            ArafdemoVO av1 = new ArafdemoVO();
            av1.setName("av" + i);
            av1.setCode("c" + i);
            list.add(av1);
        }
        return list;
    }

}
```

- 实现getJobType方法，设置作业的名称
- 实现process方法，process方法的参数为AdpKey，CommonBlock；作业的参数可以从AdpKey中获取。CommonBlock中记录了总数，分片数，处理记录的开始数与结束数。

实现Writer类

继承AbstractBlockWriter类，实现存储数据的逻辑。

```
@Component
@Slf4j
public class DbWriter extends AbstractBlockWriter<ArafdemoVO> {@Override
    public String getJobType() {
    return Constants.JOB_DB_DEMO_IMPORT;
    }

    @Override
    protected void persist(AdpKey k, ArafdemoVO v) {
        log.info("writer : {}", v);

    }


}
```

> 将Job类，Reader类，Processor类，Writer类放在同一个包中。

单元测试

> 单元测试整体框架，参加单元测试章节。单元测试

样例代码：

```
@SpringBootTest(classes = { AppServer.class })
public class OagJobTest extends ArafPgContainerTest {

    @Autowired
    private OagJob oagJob;

    @Autowired
    private MasFlightItineraryScheduleDao masFlightItineraryScheduleDao;

    @Autowired
    private MasFlightSegmentScheduleDao masFlightSegmentScheduleDao;

    @Autowired
    private ImportFileDao importFileDao;

    @Test
    @ITestDataSet(locations = { "/dataSet/mas_country.xls", "/dataSet
/mas_currency.xls" })
    public void testJob() throws InterruptedException {
        String fileName = "dataset/oag/SSIM-201910.DAT";
        String filePath = Thread.currentThread().getContextClassLoader().
getResource(fileName).getFile();

        Map<String, String> map = new HashMap<>();
        map.put(AbstractFileImportJob.FILE_PATH, filePath);

        oagJob.run(JobContext.buildContext(Constants.JOB_OAG_IMPORT, map));

        assertFlightItinerary8171();

        assertFlightItinerary888();

        assertFlightItinerary2856();

        assertFlightSegment8171();

        assertFlightSegment888();

        assertFlightSegment2856();

        assertFlightSegment173();

        assertFlightSegmentAA();

    }
}
```

- @ITestDataSet准备测试数据
- 准备测试文件，放在test resource目录下。通过代码将文件路径准备到map中。
- buildContext方法，创建作业的上下文。将文件路径传入作业中。
- 调用run方法。
- 写断言