Smart Green Platform 개발

오프라인 배치 개발가이드

문서번호 입력

Ver. 1.0

관리부서 : 통합팀

개 정 이 력

버전	작성일	변경내용[1]	작성자	승인자
1.0.0	2013-11-28	기존 별첨 10.오프라인 배치 개발가이드에서 이관. 플랫폼 개발환경 기준으로 내용 수정	최윤종	

[1] 변경 내용: 변경이 발생되는 위치와 변경 내용을 자세히 기록(장/절과 변경 내용을 기술한다.)

목 차

1. 개요

Smart Green Platform 은 배치성 업무 개발을 위해 SGP Enterprise 영역에서 배치 Framework 을 제공한다. 배치 Framework 는 배치업무를 구현함에 있어서 공통적으로 필요한 스케줄링 기능, 배치 프로세스의 관리 기능, 대량 데이터 처리 메커니즘등을 제공한다.

본 문서는 Smart Green Platform 의 배치 업무 개발 담당자가 SGP Enterprise 에서 제공하는 배치 Framework 의 구조를 이해하고, 이를 기반으로 설계 및 개발 업무를 수행하는 데 필요한 내용들을 기술하는 것을 목적으로 한다.

본 문서에서 기술하는 내용으로는
- 배치 S/W 아키텍처 구성
- 배치 개발표준
- 요소별 개발가이드

등으로 구성된다.

1.1 용어 정리

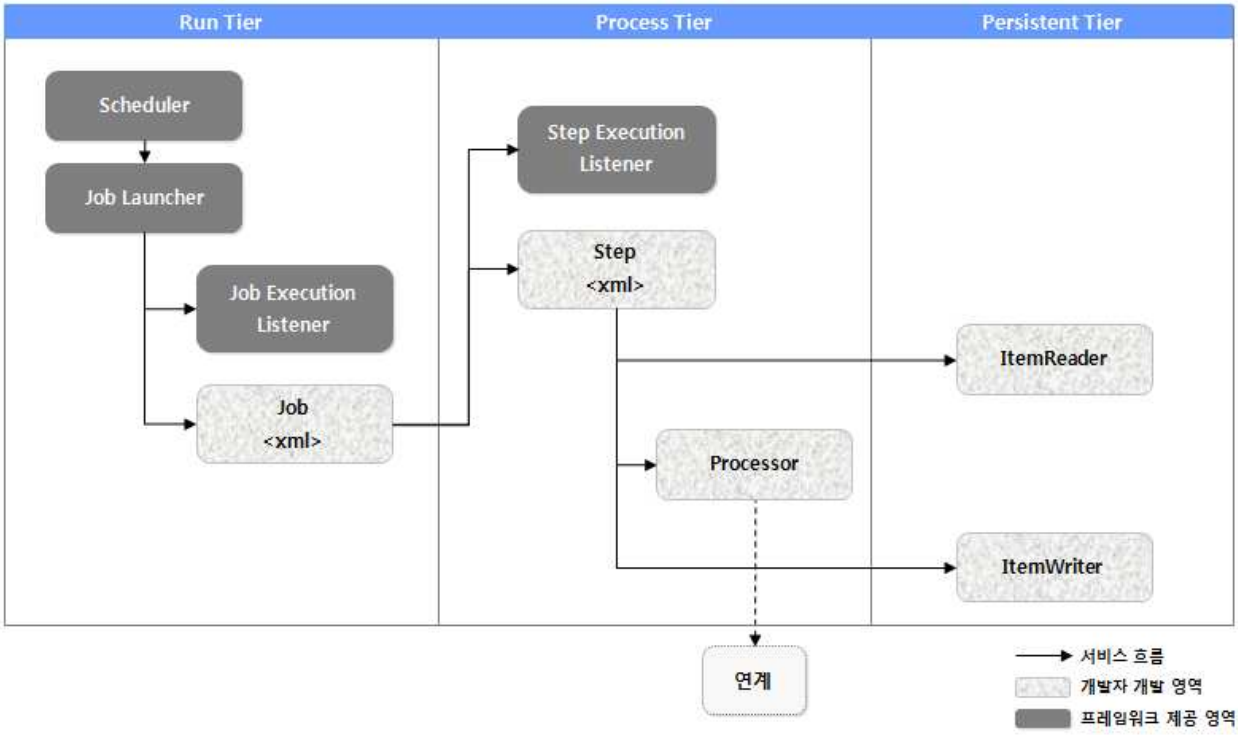
간략하게 용어를 정리하고, 자세한 개발 방법은 요소별 개발 방법을 참조하도록 한다.

No	용어	설명	비고
1	Run Tier	배치 실행에 필요한 작업을 수행함	
2	Process Tier	배치 업무 로직을 수행함	
3	Persistent Tier	배치 업무 수행을 위한 데이터를 관리함	

2. S/W 아키텍처 구성

2.1 배치 Conceptual 뷰

Conceptual 뷰는 배치 시스템의 상위 수준 Element 와 이들 사이의 관계를 정의한다. Smart Green Platform 의 배치 아키텍처를 구성하고 있는 Element 는 다음과 같다.



2.1.1 Scheduler

- 단위 배치업무의 스케줄링을 관리

2.1.2 Job Launcher

- 단위 배치업무를 실행시키고 Job Context 를 통해 실행된 Job 을 관리

2.1.3 Job

- 단위 배치업무, Step 의 Container

2.1.4 Job Execution Listener

- Job 의 수행 전/후 이벤트를 관리

2.1.5 Step

- JOB 을 구성하는 개별 업무 로직 단위

2.1.6 Step Execution Listener

- Step 의 수행 전/후 이벤트를 관리

2.1.7 ItemReader

- DB/FILE 등의 매체에서 INPUT DATA 를 읽어들임

2.1.8 Processor

- INPUT DATA 를 가공 혹은 Validate 하는 등의 비즈니스 로직을 통해 OUTPUT DATA 를 만들어 냄

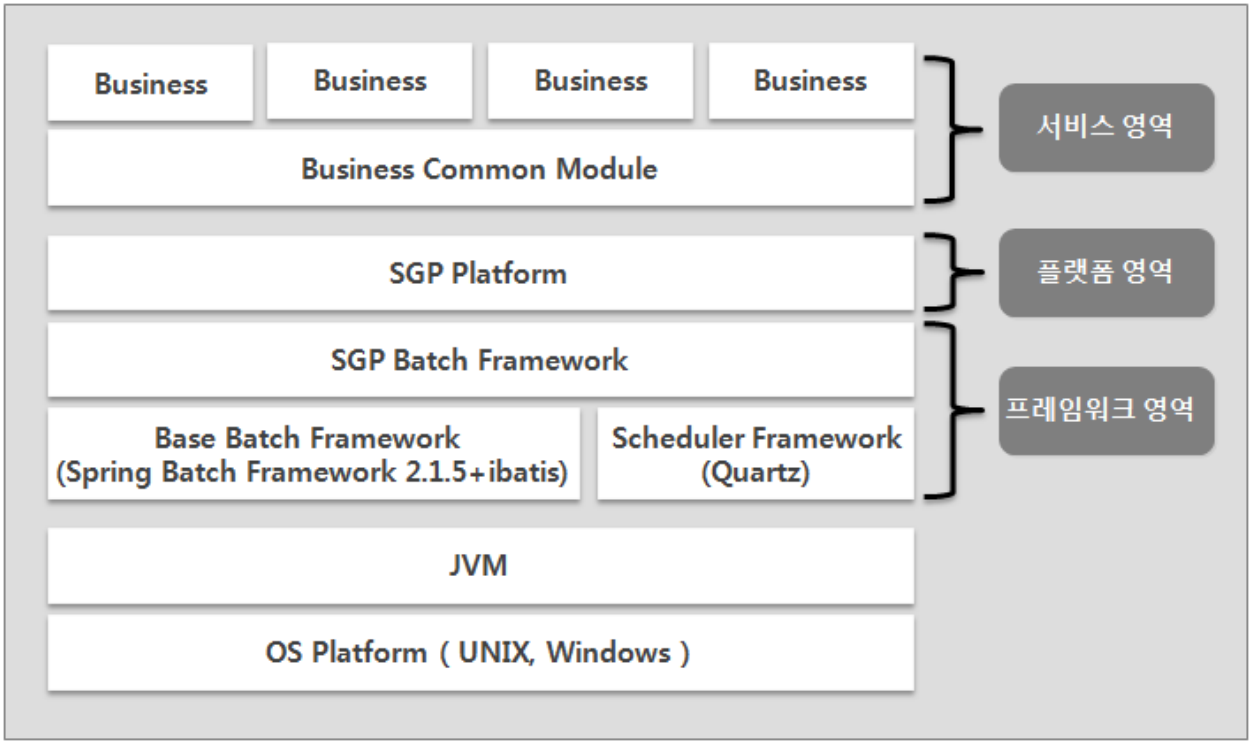
2.1.9 ItemWriter

- DB/FILE 등의 매체로 OUTPUT DATA 를 씀

2.2 배치 Layer 뷰

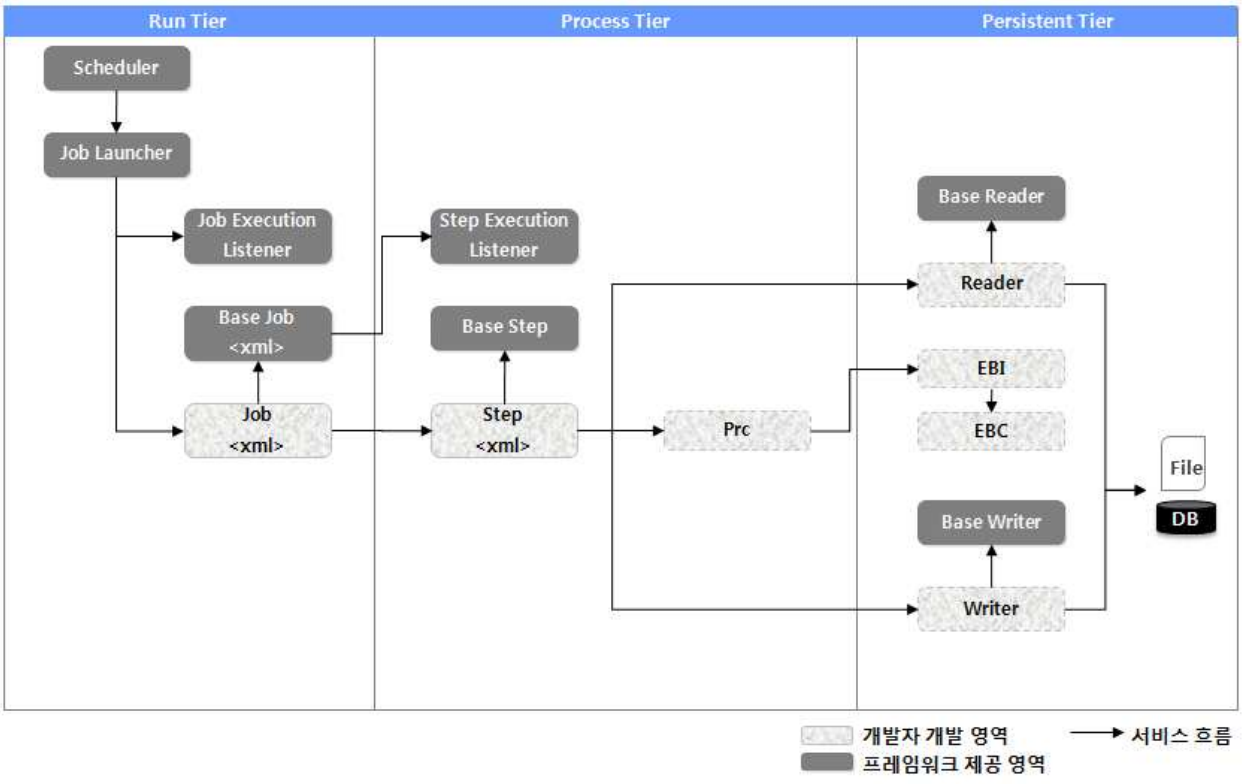
Smart Green Platform 의 배치 어플리케이션 구조를 계층적으로 표현한다.

- 오픈소스인 Spring Batch Framework 2.1.5 를 기반으로 SGP Batch Framework 를 개발한다.
- 배치 스케줄러로는 오픈소스인 Quartz 를 사용한다.
- SGP Batch Framework 를 기반으로 플랫폼 모듈을 개발한다.
- 플랫폼 모듈을 기반으로 서비스 영역을 개발한다.
- 비즈니스 응용 프로그램은 비즈니스 공통모듈을 공유한다.



2.3 개발자 개발 뷰

Spring Batch Framework 의 기본 개념을 수용하여 전체 흐름은 Job->Step->Reader->Processor->Writer 의 순서로 진행된다.



2.4 개발 구성요소

- Job, Step, Processor, Reader, Writer 는 기본 구성요소이다.
- 프레임워크에서는 기본 기능을 구현한 Base Job, Step, Reader, Writer 를 제공한다.
- 기본 기능으로 구현이 가능한 업무의 경우, XML 설정파일에 Job, Step 을 선언하고 base Reader 와 Writer 에서 사용할 query 작성, 비즈니스 로직을 담고 있는 Processor 와 데이터 모델 객체를 만든다.
- 프레임워크에서 제공하는 Base Reader, Writer 의 확장 기능이 필요할 경우에는, 각 업무 요구사항을 반영한 Reader, Writer 를 각각 개발한다.

Layer	구성요소	설명	비고
Run	Schedule	배치실행 주기 설정	XML 설정
	Job	독립적인 배치 업무 단위 여러 Step 의 조합으로 이루어짐	XML 설정 baseJob 상속
Process	Step	Job 의 흐름을 구성하는 개별 업무단위 트랜잭션에 독립적이어야함 한 Step 내부 트랜잭션의 commit 단위 지정	XML 설정 Reader/Writer 유형별 baseStep 상속
	Prc	비즈니스 로직 Flow Control(PBC 호출)	Optional
Persistent	Reader	Step 단위 처리대상 INPUT Data 를 읽어들임	매체유형별 base Reader 제공
	Writer	Step 단위 처리대상 OUTPUT Data 를 기록함	매체유형별 base Writer 제공
	EBI	Entity Bean Interface - DBIO 처리 Interface	
	EBC	Entity Bean Component - DBIO 처리 Interface 인 EBI 구현체 - AbstractSGPDao 상속 받아 구현 - @Repository 선언	

3. 개발표준

3.1 개요

개발표준은 온라인 프로그램 개발의 개발표준과 동일함을 기본 원칙으로 한다. 단, 배치 개발에서 달라지는 부분에 대해서는 본 문서에 별도로 기술한다.

3.2 업무코드

모델링 담당자가 제공하는 모델링 가이드에 따른다.

3.3 폴더구조

개발환경 구성가이드에 따른다.

3.4 모듈 기본구조

기본 폴더				용도
{프로젝트 모듈}	/src			java 프로그램 소스
	/resources	/META-INF	/jobs	Job 설정 파일
			/properties	프로퍼티 설정파일
			/schedule	Job 의 스케줄 설정파일
			/sqlmap_db2	iBATIS SQL Map xml 파일(DB2)
			/sqlmap_mssql	iBATIS SQL Map xml 파일(MSSQL)
			/sqlmap_oracle	iBATIS SQL Map xml 파일(Oracle)
			/sqlmap_postgre	iBATIS SQL Map xml 파일(Postgre)

3.5 java 패키지 구조

java 소스 파일의 패키지 구조는 업무코드명과 컴포넌트 구분에 따라 아래와 같이 한다.

com.lgcns.sgs.batch.[업무 코드명].[1 레벨 업무명].[2 레벨 업무명].[컴포넌트 구분].*.java (컴포넌트 구분: dao, model, processor, util 등)

예시)
com.lgcns.sgs.batch.dvd.psm.cmmn.dao.XXXDao.java
com.lgcns.sgs.batch.dvd.psm.cmmn.dao.XXXDaoImpl.java
com.lgcns.sgs.batch.dvd.psm.datacolct.processor.XXXPrc.java

3.6 프로퍼티 파일 경로

프로퍼티 파일은 별도의 하위 폴더를 만들지 않으며 업무별로 파일만 분리한다.

3.7 Schedule xml 파일 경로

Schedule xml 파일 경로는 업무에 따라 아래와 같이 한다.

resources/META-INF/schedule/[업무 코드명]/[1 레벨 업무명]/*.xml

3.8 Job xml 파일 경로

Job xml 파일 경로는 업무에 따라 아래와 같이 한다.

resources/META-INF/jobs/[업무 코드명]/[1 레벨 업무명]/*.xml

3.9 iBATIS SQL Map xml 파일 경로

iBatis SQL Map xml 파일 경로는 업무에 따라 아래와 같이 한다. 테스트용 iBatis SQL Map xml 파일도 동일하게 작성한다.

resources/META-INF/sqlmap/[업무 코드명]/[1 레벨 업무명]/*.xml

3.10 명명규칙

java 파일명을 제외하고 모든 파일 이름은 소문자로 시작한다. 두 개 이상의 단어가 연결될 경우에는 '_' 를 사용하지 말고, Camel Case 기법을 사용한다. 즉, 두 번째 단어부터 각 단어의 첫 글자만 대문자로 작성한다. 폴더 구분 없이 업무별로 파일만 구분되는 파일의 경우는 가급적 2 레벨 패키지 명을 파일명으로 사용하도록 한다.

3.10.1 Schedule xml 파일명

Schedule xml 파일은 2 레벨 업무별로 파일을 생성하며 파일명은 '[2 레벨 업무명]Schedule.xml 로 한다.

3.10.2 Job xml 파일명

Job xml 파일은 2 레벨 업무별로 파일을 생성하며 파일명은 '[2 레벨 업무명]Job.xml 로 한다.

3.10.3 iBATIS SQL Map xml 파일명

iBATIS SQL Map xml 파일은 EBC 당 파일을 생성하며, 파일명은 '[EBC 명]Query.xml'로 한다.

3.10.4 iBATIS SQL Map namespace / query id

iBATIS SQL Map xml 파일내의 namespace 명은 파일명과 동일하게 '[EBC 명]'으로 작성한다. query id 는 '[동사(insert/update/delete/select)][명사]'로 작성하며, EBC 의 메소드명과 동일하다.

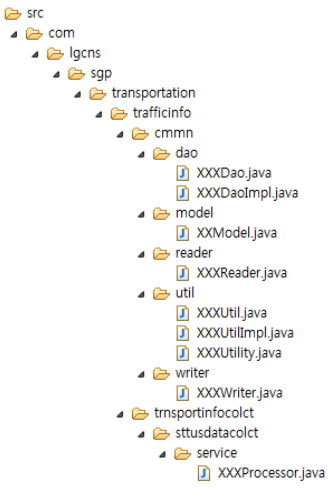
3.10.5 java 메소드명

java 메소드명은 '[의미있는 동사][명사]'의 형태로 작성한다.

3.10.6 java 파일명

java 파일명은 첫글자를 반드시 영문 대문자로 시작하며, '[의미있는 이름][컴포넌트 postfix]'의 형태로 작성한다. 컴포넌트 postfix 는 Dao, Service, Processor, Util 등과 같이 특성이 반영된 문자열을 의미한다.

어플리케이션 개발시 작성하는 java 파일의 명명 규칙은 다음과 같다.

구분(Layer)	하위 구분	패키지 이름	컴포넌트 Postfix	예제
Process	Processor	processor	Processor	
	CBI	cmmn.util	Util	
	CBC	cmmn.util	UtilImpl	
Persistent	Reader	cmmn.reader	Reader	
	Writer	cmmn.writer	Writer	
	EBI	cmmn.dao	Dao	
	EBC	cmmn.dao	DaoImpl	
Model	-	cmmn.model	Model	

3.11 설정 표준

3.11.1 프로퍼티 설정

프로퍼티 설정은 업무별로 관리하는 것을 표준으로 하며, 프로퍼티 값의 중복 가능성을 제거하기 위해 업무 코드명을 명시한다. 프로퍼티명은 '_'없이 소문자로 표기하고, 의미구분이 필요한 경우 '.'으로 구분한다.

[예시]

```
[업무 코드명].[1 레벨 업무명].[프로퍼티명]=[프로퍼티값]
dvd.psm.log.deletePeriod=-30
```

3.11.2 Bean 설정

Bean 설정의 변경 및 추가는 기존 Spring xml 파일을 수정하여 적용하는 것을 표준으로 한다. Bean id 는 인터페이스명의 첫글자를 소문자로 바꾼 것을 기본으로 하며, 중복이나 의미구분이 필요한 경우 인터페이스명 앞에 추가로 명시한다.

[예시]

```
(인터페이스명이 TransactionManager 인 경우)
<bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager" />
```

업무별로 사용범위가 한정되는 Bean id 는 업무 코드명을 명시한다.

[예시]

```
[업무 코드명].[의미구분][인터페이스명]
<bean id="dvd.psmDayDataColctJobTrigger"
 class="org.springframework.scheduling.quartz.CronTriggerBean"
 />
```

4. 엘리먼트별 개발가이드

4.1 개요

개발 뷰(Development View) 상에 존재하는 각 엘리먼트에 대한 개발방법을 가이드 한다.

4.2 Job

4.2.1 역할

독립된 배치 업무 단위를 구분한다.

4.2.2 작성방법

baseJob 을 상속받으며, spring batch 의 namespace 를 정의한다.

4.2.3 예시

```
<job id="xxx.sampleJob" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
  ...
</job>
```

4.3 Schedule

4.3.1 역할

Job 의 스케줄링을 관리한다.

4.3.2 작성방법

Scheduler.xml 에서 invoke 할 Trigger 와 JobDetail 을 정의한다. Job 별로 수행스케줄을 지정하는 Trigger 와 Job 의 수행옵션을 지정하는 JobDetail 이 한쌍으로 구성된다. cron expression 의 각각의 필드는 다음을 나타낸다.(왼쪽 -> 오른쪽 순)

필드 이름	허용 값	허용된 특수 문자
Seconds	0 ~ 59	, - * /
Minutes	0 ~ 59	, - * /
Hours	0 ~ 23	, - * /
Day-of-month	1 ~ 31	, - * ? / L W
Month	1 ~12 or JAN ~ DEC	, - * /
Day-Of-Week	1 ~ 7 or SUN-SAT	, - * ? / L #
Year (optional)	empty, 1970 ~ 2099	, - * /

< Cron Expression 의 특수문자 >

'*' : 모든 수를 나타냄. 분의 위치에 * 설정하면 *"매 분 마다"* 라는 뜻.

'?' : day-of-month 와 day-of-week 필드에서만 사용가능. 특별한 값이 없음을 나타냄

'-' : 기간을 설정. 시간 필드에 "10-12"입력하면 *"10, 11, 12 시에 동작하도록 설정"* 이란 뜻.

',' : 특정 시간을 설정. "MON,WED,FRI" 이면 *" '월,수,금' 에만 동작"* 이란 뜻.

'/' : 증가를 표현함. 초 단위에 "0/15"로 세팅 되어 있다면 *"0 초 부터 시작하여 15 초 이후에 동작"* 이란 뜻.

'L' : day-of-month 와 day-of-week 필드에만 사용하며 마지막날을 나타냄. 만약 day-of-month 에 "L" 로 되어 있다면 이번 달의 마지막에 실행하겠다는 것을 나타냄.

'W' : day-of-month 필드에만 사용되며, 주어진 기간에 가장 가까운 평일(월~금)을 나타냄. 만약 "15W" 이고 이번 달의 15 일이 토요일이라면 가장가까운 14 일 금요일날 실행됨. 또 15 일이 일요일이라면 가장 가까운 평일인 16 일 월요일에 실행되게 됨. 만약 15 일이 화요일이라면 화요일인 15 일에 수행됨.

"LW" : L 과 W 를 결합하여 사용할 수 있으며 "LW"는 *"이번달 마지막 평일"*을 나타냄

"#" : day-of-week 에 사용됨. "6#3" 이면 3(3)번째 주 금요일(6) 이란 뜻. 1 은 일요일 ~ 7 은 토요일

< Sample >

Expression	Meaning
"0 0 12 * * ?"	매일 12 시에 실행
"0 15 10 ? * *"	매일 10 시 15 분에 실행
"0 15 10 * * ?"	매일 10 시 15 분에 실행
"0 15 10 * * ? *"	매일 10 시 15 분에 실행
"0 15 10 * * ? 2010"	2010 년 동안 매일 10 시 15 분에 실행
"0 * 14 * * ?"	매일 14 시에서 시작해서 14:59 분 에 끝남
"0 0/5 14 * * ?"	매일 14 시에 시작하여 5 분 간격으로 실행되며 14:55 분에 끝남
"0 0/5 14,18 * * ?"	매일 14 시에 시작하여 5 분 간격으로 실행되며 14:55 분에 끝나고, 매일 18 시에 시작하여 5 분간격으로 실행되며 18:55 분에 끝난다.
"0 0-5 14 * * ?"	매일 14 시에 시작하여 14:05 분에 끝난다.

작성한 Schedule Trigger 는 resources/META-INF/schedule 폴더에 있는 schedule-[업무코드명]-triggers.xml 에 업무별로 등록해주어야 한다.

4.3.3 예시

[xxxSchedule.xml]

```
<bean id="xxx.sampleJobTrigger" parent="baseJobTrigger">
        <property name="jobDetail" ref="sampleJobDetail" />
        <property name="cronExpression" value="0/10 * * * * ?" />
</bean>

<bean id="xxx.sampleJobDetail" parent="baseJobDetail">
        <property name="jobDataAsMap">
            <map>
                <entry key="jobName" value="xxx.sampleJob" />
                <entry key="jobLocator" value-ref="jobLocator"/>
                <entry key="jobLauncher" value-ref="jobLauncher"/>
            </map>
        </property>
</bean>
```

[schedule-xxx-context.xml]

```
<bean id="xxx.triggerRegister" class="com.lgcns.sgp.framework.batch.schedule.TriggerRegister">
    <property name="triggers">
        <list>
            <ref bean="xxx.sampleJobTrigger" />
            ...
        </list>
    </property>
</bean>
```

4.4 Step

4.4.1 역할

Job 을 구성하는 독립적으로 구분되는 업무단위이며, 트랜잭션은 하나의 Step 내에서 보장된다.

4.4.2 작성방법

각 Step 파일에서는 baseJob.xml 파일을 import 한다.

Step 의 기본 자원(commit 단위, reader/writer 의 유형등)에 대한 속성을 정의한 상위 Step 이 제공된다. 각 유형별 상위 Step 을 상속받아서 업무 Step 을 정의한다. 상위 Step 의 자원 속성은 개별 Step 에서 override 가능하다.

제공되는 Step 의 종류는 다음과 같다.

- baseDBtoDBStep
  -DB 에서 input 데이터를 읽어들여 가공후 DB 에 다시 write 하는 기본 패턴 지원
  -default 로 100 건의 데이터 단위로 commit 하도록 설정되어 있다.

- baseDBtoNullWriterStep
  -DB 에서 input 데이터를 읽어들여 가공후 write 작업없이 종료되는 경우
  -default 로 100 건의 데이터 단위로 commit 하도록 설정되어 있다.

- baseTaskStep
  -Reader/Writer 의 작업 구분 없이 단일 작업수행으로 완료되는 경우

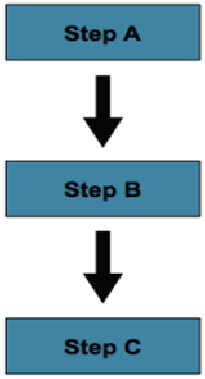
4.4.3 예시

```
<import resource="classpath:/META-INF/jobs/baseJob.xml" />

<job id="xxx.pattern1_1Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="xxx.sampleDBtoDBStep" parent="baseDBtoDBStep">
        <tasklet>
            <chunk reader="xxx.samplePagingReader" processor="Pattern1_1Processor"
                    writer="xxx.sampleBatchWriter"/>
        </tasklet>
    </step>
</job>

<job id="xxx.pattern1_2Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="xxx.singleTaskStep" parent="baseDBtoNullWriterStep">
        <tasklet>
            <chunk reader="xxx.samplePagingReader" processor="xxx.pattern1_1Processor"/>
        </tasklet>
    </step>
</job>


<job id="xxx.pattern2_1Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="singleTaskStep" parent="baseTaskStep">
        <tasklet ref="xxx.pattern2_1Processor"/>
    </step>
</job>
```

4.4.4 Step Flow Control

하나의 Job 에서 여러 개의 Step 을 선언하는 것이 가능하다. 이경우, Step 간의 Flow 를 제어하기 위해서 다음과 같은 문법이 적용된다.

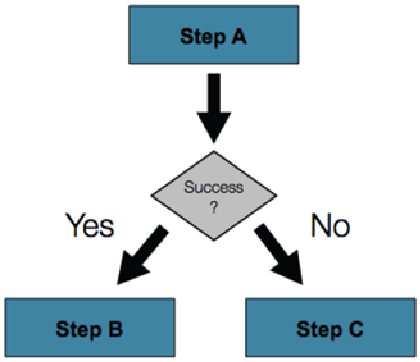
4.4.4.1 Sequential Flow




```
<job id="job">
    <step id="stepA" parent="s1" next="stepB" />
    <step id="stepB" parent="s2" next="stepC" />
    <step id="stepC" parent="s3" />
</job>
```

stepA, stepB, stepC 가 순서대로 수행된다. 수행중 오류가 발생하면 JOB 수행이 중지된다.

4.4.4.2 Conditional Flow




```
<job id="job">
    <step id="stepA" parent="s1">
        <next on="*" to="stepB" />
        <next on="FAILED" to="stepC" />
    </step>
    <step id="stepB" parent="s2" next="stepC" />
    <step id="stepC" parent="s3" />
</job>
```

stepA, stepB, stepC 가 순서대로 수행된다. 단, stepA 수행중 오류가 발생하면 stepC 가 수행된다.

4.4.5 Step 간 데이터 공유

하나의 Job 에 속한 Step 간에 데이터를 공유할 수 있다. 단, 메모리에 상주하게 되므로, 단순 KEY VALUE 데이터만으로 제한한다. (저장 KEY 는 "key"로 고정됨)

[데이터를 저장할 경우]

```
public void onProcess(){

    this.setStepContextData("key", data);
```

[데이터를 얻어올 경우]

```
public void onProcess(){

    Object key = this.getStepContextData("key");
```

4.4.6 Step 순서 변경

Job 에 속한 Step 은 Job 설정파일에 정의되며, 이때 Job 내부 Step 의 실행순서도 명시적으로 정의된다. 기본적으로 Step 의 실행순서 변경은 개발자에 의해 설정파일을 update 하는 것이 원칙이나, 특정 업무 요건상 온라인 사용자에 의한 설정변경이 필요한 경우에는 다음의 가이드를 따른다. 단, 설정값을 변경한 후 배치서버 재시작 시점에 반영된다. 런타임에는 실시간으로 변경된 설정값이 적용되지 않는다.

- 온라인 프로그램의 서비스에서 BatchJobManagerAdapter 의 updateStepOrder 를 호출한다. 파라메터는 JobName 과 순서별 Step 명이며, 순서별 Step 명은 컴마(,)를 구분자로 한다.

- 변경된 Step 순서의 확인은 실행중인 배치 데몬의 Job 설정파일에서 Update 여부를 직접 확인한다.

[updateStepOrder 호출 예시]

```
import com.lgcns.sgp.framework.batch.remote.adapter.BatchJobManagerAdapter;

public class XxxServiceImpl{
```

```
@Autowired
BatchJobManagerAdapter batchJobManagerAdapter;

public void serviceOperation() {
    batchJobManagerAdapter.updateStepOrder("JobName", "step3,step2,step1,step4");
```

4.5 Reader

4.5.1 역할

파일, DB 등의 매체로부터 INPUT 데이터를 읽어들인다. 각 매체 유형별로 최적화된 기본 Reader 를 제공한다.

4.5.2 작성방법

유형별 상위 Reader 를 상속받는다. 제공되는 Reader 의 종류는 다음과 같다.

- baseDBPagingReader
  -작성한 Query 문의 ID 만 설정해주면 기본 commit 단위인 100 건씩 데이터를 조회해온다.
  -Query 작성시는 페이징을 위해 다음 형식을 따른다.

업무유형의 특성에 따라 별도의 Reader 를 만들어 사용할 수도 있다.

4.5.3 예시

```
<bean id="xxx.samplePagingReader" parent="basePagingDBReader">
    <property name="queryId" value="Pattern1_1.selectSampleList" />
</bean>
```

4.6 Writer

4.6.1 역할

파일/DB 등의 매체에 OUTPUT Data 를 Write 한다. 각 매체 유형별로 최적화된 기본 Writer 가 제공된다.

4.6.2 작성방법

유형별 상위 Writer 를 상속받는다. 제공되는 Writer 의 종류는 다음과 같다.

- baseDBWriter
  -주어진 Query 를 iBATIS Framework 의 Batch 처리 형식으로 수행한다.
- multipleWriter
  -item 에서 지정한 writerId 에 따라 writer 를 실행한다.

4.6.3 예시

```
<bean id="xxx.sampleBatchWriter" parent="baseDBWriter">
    <property name="statementId" value="Pattern1_1.updateSample" />
</bean>
```

```
<bean id="xxx.sampleBatchMultiWriter" parent="multipleWriter">
    <property name="writerMap" >
        <entry key="Table1.writerId" value-ref="xxx.table1BatchWriter" />
        <entry key="Table2.writerId" value-ref="xxx.table2BatchWriter" />
    </property>
</bean>

<bean id="xxx.table1BatchWriter" parent="baseDBWriter">
    <property name="statementId" value="Pattern3_1.updateTable1" />
</bean>

<bean id="xxx.table2BatchWriter" parent="baseDBWriter">
    <property name="statementId" value="Pattern3_1.updateTable2" />
</bean>
```

4.7 Processor

4.7.1 역할

Reader 로부터 넘겨받은 INPUT Data 를 가공, 검증하여 Writer 로 전달하는 경우와 Reader/Writer 없이 단일 로직 처리만 수행하는 경우가 있다.

4.7.2 작성방법

유형별 상위 Processor 를 상속받아 onProcess 함수를 구현한다.

- BaseProcessor
  - Reader/Writer 구조인 일반적인 경우
- BaseTaskProcessor
  - 단일로직처리만 수행하는 경우
- MultipleWriterProcessor
  - MultipleWriter 를 사용하는 경우

4.7.3 예시 - Prc (xml)

```
<bean id="xxx.pattern1_1Processor"
    class="com.lgcns.sgs.batch.sample.dbpattern.pattern1.processor.Pattern1_1Processor"
    scope="step" />
```

4.7.4 예시 - Prc (class)

```
# BaseProcessor
public class Pattern1_1Processor extends BaseProcessor<Sample, Sample> {

    private Sample failedItem = null;

    @Autowired
    private Pattern1_1Dao dao;

    public Sample onProcess(Sample item){

        if("N".equals(item.getFlag())){
            item.setFlag("Y");
            if("A".equals(item.getPGroup())){
```

```
                dao.insertSampleA(item);
            }else if("B".equals(item.getPGroup())){
                dao.insertSampleB(item);
            }else
                failedItem = item;
        }

        if (failedItem != null && failedItem.equals(item)) {
            failedItem = item;
            throw new SGPServiceException("Some bad data for " + failedItem);
        }
        return item;
    }
}
```

 # TaskProcessor

```
public class Pattern2_1Processor extends BaseTaskProcessor{

    @Autowired
    private Pattern2_1Dao dao;

    @Override
    public void onProcess(){
        dao.insertItem();
    }
}
```

 # MultipleWriterProcessor

```
public class SampleMultipleWriterProcessor extends MultipleWriterProcessor<ChunkModel>{

    @Override
    public List<MultipleWriterItem> onProcess( ChunkModel item ) {

        int chunkId = item.getChunkId();
        int chunkValue = item.getChunkValue();

        List<MultipleWriterItem> outItem = new ArrayList<MultipleWriterItem>();

        // updateChunkWriter로 보낼 item 생성
        ChunkModel chunkItem = new ChunkModel();
        chunkItem.setChunkId( chunkId );
        chunkItem.setChunkValue( chunkValue + 1 );

        outItem.add( new MultipleWriterItem( "updateChunkWriter", chunkItem ) );
```

```
        // updateDummyWriter로 보낼 item 생성
        for( int idx = 0; idx < 3; idx++ ) {
            DummyModel dummyItem = new DummyModel();
            dummyItem.setDummyId( chunkId + "_" + idx );
            dummyItem.setDummyValue( chunkValue );

            outItem.add( new MultipleWriterItem( "updateDummyWriter", dummyItem ) );
        }

        return outItem;
    }
}
```

※ Dao 와 SQL Map 파일 작성은 온라인 개발표준과 동일하다.

5. 패턴별 예시

5.1 Chunk Process

Reader 에서 chunk 단위로 item 을 읽어와 Processor 에서 가공 후 Writer 에서 write 하는 패턴이다. 예시는 Writer 가 최종적으로 write 하는 대상의 종류 수에 따라 Single Writer, Multi Writer, Null Writer 패턴으로 분류한다.

5.1.1 Single Writer Process

하나의 Writer 에서 item 을 write 한다. Writer 는 chunk 단위로 write 한다.

[Job.xml 예시]

```
<job id="xxx.pattern1_1Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="xxx.p1_1SampleDBtoDBStep" parent="baseDBtoDBStep">
        <tasklet>
            <chunk reader="xxx.p1_1SamplePagingReader" processor="xxx.p1_1SampleBatchProcessor"
writer="xxx.p1_1SampleBatchWriter"/>
        </tasklet>
    </step>
</job>

<bean id="xxx.p1_1SamplePagingReader" parent="basePagingDBReader">
```

```
    <property name="queryId" value="pattern1_1.selectPagingSampleBatchFirstList" />
</bean>

<bean id="xxx.p1_1SampleBatchWriter" parent="baseDBWriter">
    <property name="statementId" value="pattern1_1.updateSampleBatchSecond" />
</bean>

<bean id="xxx.p1_1SampleBatchProcessor"
    class="com.lgcns.sgs.sample.batch.pattern1_1.processor.P1_1SampleChunkProcessor"
    scope="step" />
```

[Query.xml 예시]

```
<typeAlias alias="p1_1SampleBatchFirstModel"
type="com.lgcns.sgs.sample.batch.pattern1_1.cmmn.model.P1_1SampleBatchFirstModel"/>
<typeAlias alias="p1_1SampleBatchSecondModel"
type="com.lgcns.sgs.sample.batch.pattern1_1.cmmn.model.P1_1SampleBatchSecondModel"/>

<!-- chunk단위로 샘플모델1의 목록을 조회한다. -->
<select id="selectPagingSampleBatchFirstList" resultClass="p1_1SampleBatchFirstModel">
<include refid="common.pagingPrefix" />
<![CDATA[
    SELECT ROW_NUMBER() OVER (ORDER BY SAMPLE_ID) RNUM,
           SAMPLE_ID        AS sampleId,
           SAMPLE_VALUE    AS sampleValue
      FROM SAMPLE_BATCH_FIRST
]]>
 <include refid="common.pagingPostfix" />
</select>

 <!-- 샘플모델2의 sampleValue를 변경한다. -->
<update id="updateSampleBatchSecond" parameterClass="p1_1SampleBatchSecondModel">
<![CDATA[
UPDATE SAMPLE_BATCH_SECOND SET SAMPLE_VALUE = #sampleValue# WHERE SAMPLE_ID = #sampleId#
  ]]>
</update>
```

[Processor 예시]

```
/**
 * <pre>
 * 패턴1-1의 Processor
 * chunk 단위로 읽어들여 item 단위로 processor를 실행한다.
 * </pre>
 */
```

```
public class P1_1SampleChunkProcessor extends BaseProcessor<P1_1SampleBatchFirstModel,
P1_1SampleBatchSecondModel> {

    /**
     * 샘플모델1의 sampleValue를 1증가하여 샘플모델2로 전달한다.
     * (reader로부터 받은 item을 writer에 전달하기 전에 가공한다.)
     *
     * @param inItem reader로부터 받은 item
     * @return writer로 전달할 item
     * @see com.lgcns.sgp.framework.batch.processor.BaseProcessor#onProcess(java.lang.Object)
     */
    public P1_1SampleBatchSecondModel onProcess( P1_1SampleBatchFirstModel inItem ) {

        P1_1SampleBatchSecondModel outItem = new P1_1SampleBatchSecondModel();
        outItem.setSampleId( inItem.getSampleId() );
        outItem.setSampleValue( inItem.getSampleValue() + 1 );

        if( inItem.getSampleId() == 10 ) {
            // skip하려고 할때는 null로 전달
            outItem = null;
        }

        return outItem;
    }
}
```

5.1.2 Multi Writer Process

다수의 Writer 에서 item 을 write 한다. Reader 가 읽어 들인 item 은 Processor 에서 writerId 와 item 으로 매핑된 MultipleWriterItem 으로 가공되고 MultipleWriter 에서 writerId 에 해당하는 writer 에 각각 item 을 전달하여 chunk 단위로 write 한다.

[Job.xml 예시]

```
<job id="xxx.pattern1_2Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="xxx.p1_2SampleDBtoDBStep" parent="baseDBtoDBStep">
        <tasklet>
            <chunk reader="xxx.p1_2SamplePagingReader" processor="xxx.p1_2SampleBatchProcessor"
writer="xxx.p1_2SampleMultipleWriter"/>
        </tasklet>
    </step>
</job>
```

```
<bean id="xxx.p1_2SamplePagingReader" parent="basePagingDBReader">
    <property name="queryId" value="pattern1_2.selectPagingSampleBatchFirstList" />
</bean>

<bean id="xxx.p1_2SampleMultipleWriter" parent="multipleWriter">
    <property name="writerMap">
        <map>
            <entry key="p1_2FirstWriterId" value-ref="xxx.p1_2SampleBatchFirstWriter" />
            <entry key="p1_2SecondWriterId" value-ref="xxx.p1_2SampleBatchSecondWriter" />
        </map>
    </property>
</bean>
<bean id="xxx.p1_2SampleBatchFirstWriter" parent="baseDBWriter">
    <property name="statementId" value="pattern1_2.updateSampleBatchFirst" />
</bean>
<bean id="xxx.p1_2SampleBatchSecondWriter" parent="baseDBWriter">
    <property name="statementId" value="pattern1_2.updateSampleBatchSecond" />
</bean>

<bean id="xxx.p1_2SampleBatchProcessor"
    class="com.lgcns.sgs.sample.batch.pattern1_2.processor.P1_2SampleMultiWriterProcessor"
    scope="step" />
```

[Query.xml 예시]

```
<typeAlias alias="p1_2SampleBatchFirstModel"
type="com.lgcns.sgs.sample.batch.pattern1_2.cmmn.model.P1_2SampleBatchFirstModel"/>
<typeAlias alias="p1_2SampleBatchSecondModel"
type="com.lgcns.sgs.sample.batch.pattern1_2.cmmn.model.P1_2SampleBatchSecondModel"/>

<!-- chunk단위로 샘플모델1의 목록을 조회한다. -->
<select id="selectPagingSampleBatchFirstList" resultClass="p1_2SampleBatchFirstModel">
<include refid="common.pagingPrefix" />
<![CDATA[
    SELECT ROW_NUMBER() OVER (ORDER BY SAMPLE_ID) RNUM,
           SAMPLE_ID        AS sampleId,
           SAMPLE_VALUE   AS sampleValue
      FROM SAMPLE_BATCH_FIRST
]]>
<include refid="common.pagingPostfix" />
</select>

 <!-- 샘플모델1의 sampleValue를 변경한다. -->
<update id="updateSampleBatchFirst" parameterClass="p1_2SampleBatchFirstModel">
<![CDATA[
```

```
    UPDATE SAMPLE_BATCH_FIRST SET SAMPLE_VALUE = #sampleValue# WHERE SAMPLE_ID = #sampleId#
]]>
</update>

<!-- 샘플모델2의 sampleValue를 변경한다. -->
<update id="updateSampleBatchSecond" parameterClass="p1_2SampleBatchSecondModel">
<![CDATA[
    UPDATE SAMPLE_BATCH_SECOND SET SAMPLE_VALUE = #sampleValue# WHERE SAMPLE_ID = #sampleId#
]]>
</update>
```

[Processor 예시]

```
/**
 * <pre>
 * 패턴1-2의 Processor
 * multi writer processor 샘플
 * </pre>
 */
public class P1_2SampleMultiWriterProcessor extends MultipleWriterProcessor<P1_2SampleBatchFirstModel> {

    /**
     * 샘플모델1의 sampleValue를 1 증가시키고,
     * 샘플모델2의 sampleValue를 샘플모델1의 sampleValue를 참고하여 변경한다.
     *
     * @param inItem reader로부터 받은 item
     * @return writer로 전달할 item
     * @see com.lgcns.sgp.framework.batch.processor.BaseProcessor#onProcess(java.lang.Object)
     */
    public List<MultipleWriterItem> onProcess( P1_2SampleBatchFirstModel inItem ) {

        //1-1. 최종 writer에 전달되는 모델 생성
        P1_2SampleBatchFirstModel firstModel = new P1_2SampleBatchFirstModel();
        firstModel.setSampleId( inItem.getSampleId() );
        firstModel.setSampleValue( inItem.getSampleValue() + 1 );

        //1-2. MultipleWriterItem에 writer와 모델 매핑
        MultipleWriterItem item1 = new MultipleWriterItem( "p1_2FirstWriterId", firstModel );

        //1-3. List<MultipleWriterItem>에 추가
        List<MultipleWriterItem> outItem = new ArrayList<MultipleWriterItem>();
        outItem.add( item1 );

        //2. input item 한건에 여러건의 output item 연결
        for( int idx = 0; idx < 5; idx++ ) {
```

```
            //2-1. 최종 writer에 전달되는 모델 생성
            P1_2SampleBatchSecondModel secondModel = new P1_2SampleBatchSecondModel();
            secondModel.setSampleId( inItem.getSampleId() * 5 + idx );
            secondModel.setSampleValue( inItem.getSampleValue() + idx );

            //2-2. MultipleWriterItem에 writer와 모델 매핑
            MultipleWriterItem item2 = new MultipleWriterItem( "p1_2SecondWriterId", secondModel );

            //2-3. List<MultipleWriterItem>에 추가
            outItem.add( item2 );
        }

        //3. List<MultipleWriterItem> 전달
        return outItem;
    }
}
```

5.1.3 Null Writer Process

Writer 없이 Processor 에서 배치작업이 이루어지는 경우 step 을 baseDBtoNullWriterStep 으로 설정한다. Processor 는 item 단위로 실행된다.

[Job.xml 예시]

```
<job id="xxx.pattern1_3Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="xxx.p1_3SampleDBtoNullWriterStep" parent="baseDBtoNullWriterStep">
        <tasklet>
            <chunk reader="xxx.p1_3SamplePagingReader" processor="xxx.p1_3SampleBatchProcessor" />
        </tasklet>
    </step>
</job>

<bean id="xxx.p1_3SamplePagingReader" parent="basePagingDBReader">
    <property name="queryId" value="pattern1_3.selectPagingSampleBatchFirstList" />
</bean>

<bean id="xxx.p1_3SampleBatchProcessor"
    class="com.lgcns.sgs.sample.batch.pattern1_3.processor.P1_3SampleDBtoNullWriterProcessor"
    scope="step" />
```

[Query.xml 예시]

```
<typeAlias alias="p1_3SampleBatchFirstModel"
type="com.lgcns.sgs.sample.batch.pattern1_3.cmmn.model.P1_3SampleBatchFirstModel"/>
```

```
<!-- chunk단위로 샘플모델1의 목록을 조회한다. -->
<select id="selectPagingSampleBatchFirstList" resultClass="p1_3SampleBatchFirstModel">
<include refid="common.pagingPrefix" />
<![CDATA[
    SELECT ROW_NUMBER() OVER (ORDER BY SAMPLE_ID) RNUM,
           SAMPLE_ID      AS sampleId,
           SAMPLE_VALUE   AS sampleValue
      FROM SAMPLE_BATCH_FIRST
]]>
<include refid="common.pagingPostfix" />
</select>

<!-- 샘플모델1의 sampleValue를 변경한다. -->
<update id="updateSampleBatchFirst" parameterClass="p1_3SampleBatchFirstModel">
<![CDATA[
    UPDATE SAMPLE_BATCH_FIRST SET SAMPLE_VALUE = #sampleValue# WHERE SAMPLE_ID = #sampleId#
]]>
</update>
```

[Processor 예시]

```
/**
 * <pre>
 * 패턴1-3의 Processor
 * DB to null writer processor 샘플
 * (DB에서 input 데이터를 읽어들여 가공후 write작업없이 종료되는 경우)
 * </pre>
 */
public class P1_3SampleDBtoNullWriterProcessor extends BaseProcessor<P1_3SampleBatchFirstModel,
P1_3SampleBatchFirstModel> {

    @Autowired
    P1_3SampleBatchDao sampleBatchDao;

    /**
     * 샘플모델1의 sampleValue를 1 증가된 값으로 변경한다.
     *
     * @param inItem reader로부터 받은 item
     * @return writer로 전달할 item (writer가 없으므로 inItem을 pass한다)
     * @see com.lgcns.sgp.framework.batch.processor.BaseProcessor#onProcess(java.lang.Object)
     */
    public P1_3SampleBatchFirstModel onProcess( P1_3SampleBatchFirstModel item ) {

        item.setSampleValue( item.getSampleValue() + 1 );
```

```
            sampleBatchDao.updateSampleBatchFirst( item );

            return item;
        }
}
```

5.2 Task Process

5.2.1 Single Task Process

Reader, Writer 없이 Processor 에서 모든 배치작업이 이루어지는 경우 step 을 baseTaskStep 으로 설정한다.

[Job.xml 예시]

```
<job id="xxx.pattern2_1Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="xxx.p2_1SampleTaskStep" parent="baseTaskStep">
        <tasklet ref="xxx.p2_1SampleBatchProcessor" />
    </step>
</job>

<bean id="xxx.p2_1SampleBatchProcessor"
    class="com.lgcns.sgs.sample.batch.pattern2_1.processor.P2_1SampleTaskProcessor"
    scope="step" />
```

[Query.xml 예시]

```
<typeAlias alias="p2_1SampleBatchFirstModel"
type="com.lgcns.sgs.sample.batch.pattern2_1.cmmn.model.P2_1SampleBatchFirstModel"/>
<typeAlias alias="p2_1SampleBatchSecondModel"
type="com.lgcns.sgs.sample.batch.pattern2_1.cmmn.model.P2_1SampleBatchSecondModel"/>

<!-- 샘플모델1의 전체 목록을 조회한다. -->
<select id="selectSampleBatchFirstList" resultClass="p2_1SampleBatchFirstModel">
<![CDATA[
    SELECT SAMPLE_ID        AS sampleId,
           SAMPLE_VALUE     AS sampleValue
      FROM SAMPLE_BATCH_FIRST
]]>
</select>

<!-- 샘플모델2의 sampleValue를 변경한다. -->
<update id="updateSampleBatchSecond" parameterClass="p2_1SampleBatchSecondModel">
```

```
<![CDATA[
    UPDATE SAMPLE_BATCH_SECOND SET SAMPLE_VALUE = #sampleValue# WHERE SAMPLE_ID = #sampleId#
]]>
</update>
```

[Processor 예시]

```
/**
 * <pre>
 * 패턴2-1의 Processor
 * Single Task 샘플
 * (Reader/Writer의 작업 구분 없이 단일 작업수행으로 완료되는 경우)
 * </pre>
 */
public class P2_1SampleTaskProcessor extends BaseTaskProcessor {

    @Autowired
    P2_1SampleBatchDao sampleBatchDao;

    /**
     * 샘플모델1의 전체목록을 조회하여, 샘플모델1의 sampleValue보다 1 증가된 값으로
     * 샘플모델2의 sampleValue를 변경한다.
     *
     * @see com.lgcns.sgp.framework.batch.processor.BaseTaskProcessor#onProcess(java.lang.Object)
     */
    public void onProcess() {

        List<P2_1SampleBatchFirstModel> sampleBatchFirstModelList =
sampleBatchDao.selectSampleBatchFirstList();

        for ( P2_1SampleBatchFirstModel sampleBatchFirstModel : sampleBatchFirstModelList ) {

            P2_1SampleBatchSecondModel sampleBatchSecondModel = new P2_1SampleBatchSecondModel();
            sampleBatchSecondModel.setSampleId( sampleBatchFirstModel.getSampleId() );
            sampleBatchSecondModel.setSampleValue( sampleBatchFirstModel.getSampleValue() + 1 );

            sampleBatchDao.updateSampleBatchSecond( sampleBatchSecondModel );
        }
    }
}
```

5.3 Exception Handling

5.3.1 Skip Listener

Reader, Processor, Writer 에서 예외 발생시 Skip Listener 가 우선 실행된다. 별도 처리가 없는 경우 예외가 발생한 item 은 skip 처리된다. 아래 예시는 Writer 에서 예외 발생시 item 을 수정하여 insert 한다. Skip Listener 에서 예외가 발생하는 경우, 해당 chunk 전체가 실패로 처리되어 rollback 된다.

[Job.xml 예시]

```
<job id="xxx.pattern3_1Job" parent="baseJob" xmlns="http://www.springframework.org/schema/batch">
    <step id="xxx.p3_1SampleDBtoDBStep" parent="baseDBtoDBStep">
        <tasklet>
            <!-- skip limit은 infinite이 없음 -->
            <chunk reader="xxx.p3_1SamplePagingReader" processor="xxx.p3_1SampleBatchProcessor"
writer="xxx.p3_1SampleBatchWriter" skip-limit="999999" >
                <skippable-exception-classes>
                    <include class="java.lang.Exception" />
                </skippable-exception-classes>
            </chunk>
            <listeners>
                <listener ref="xxx.p3_1SampleSkipListener" />
            </listeners>
        </tasklet>
    </step>
</job>

<bean id="xxx.p3_1SamplePagingReader" parent="basePagingDBReader">
    <property name="queryId" value="pattern3_1.selectPagingSampleBatchFirstList" />
</bean>
<bean id="xxx.p3_1SampleBatchWriter" parent="baseDBWriter">
    <property name="statementId" value="pattern3_1.insertSampleBatchSecond" />
</bean>
<bean id="xxx.p3_1SampleBatchProcessor"
    class="com.lgcns.sgs.sample.batch.pattern3_1.processor.P3_1SampleChunkProcessor"
    scope="step" />

<bean id="xxx.p3_1SampleSkipListener"
    class="com.lgcns.sgs.sample.batch.pattern3_1.listener.P3_1SampleSkipListener"
    scope="step" />
```

[Listener 예시]

```
/**
 * <pre>
 * 패턴3-1의 Skip Listener.
 * </pre>
 */
public class P3_1SampleSkipListener implements SkipListener<P3_1SampleBatchFirstModel,
P3_1SampleBatchSecondModel> {

    @Autowired
    P3_1SampleSkipRecoveryDao dao;

    /**
     * @see org.springframework.batch.core.SkipListener#onSkipInProcess(java.lang.Object, java.lang.Throwable)
     */
    @Override
    public void onSkipInProcess( P3_1SampleBatchFirstModel item, Throwable t ) {
        // processor에서 exception 발생시 사용
    }

    /**
     * @see org.springframework.batch.core.SkipListener#onSkipInRead(java.lang.Throwable)
     */
    @Override
    public void onSkipInRead( Throwable t ) {
        // reader에서 exception 발생시 사용(Callback)
    }

    /**
     * @see org.springframework.batch.core.SkipListener#onSkipInWrite(java.lang.Object, java.lang.Throwable)
     */
    @Override
    public void onSkipInWrite( P3_1SampleBatchSecondModel item, Throwable t ) {
        // writer에서 exception 발생시 사용
        item.setSampleId( item.getSampleId() + 2000 );
        item.setSampleValue( item.getSampleValue() + 10000 );

        //재가공한 item을 저장한다. dao가 실패할 경우, chunk 전체가 fail이 된다.
        dao.insertSampleBatchSecond( item );
    }

}
```