

# 전자정부 표준프레임워크 배치 실행환경



1. 개요
2. Batch 구성요소
3. Spring Batch 2.x
4. Spring Batch 3.x
5. Batch Processing
6. Batch Support
7. eGovFrame Batch 제공기능
8. 참고자료

## 1. 개요

- 실행환경 배치처리 레이어
- Batch 개념
- Spring Batch

## 2. Batch 구성요소

## 3. Spring Batch 2.x

## 4. Spring Batch 3.x

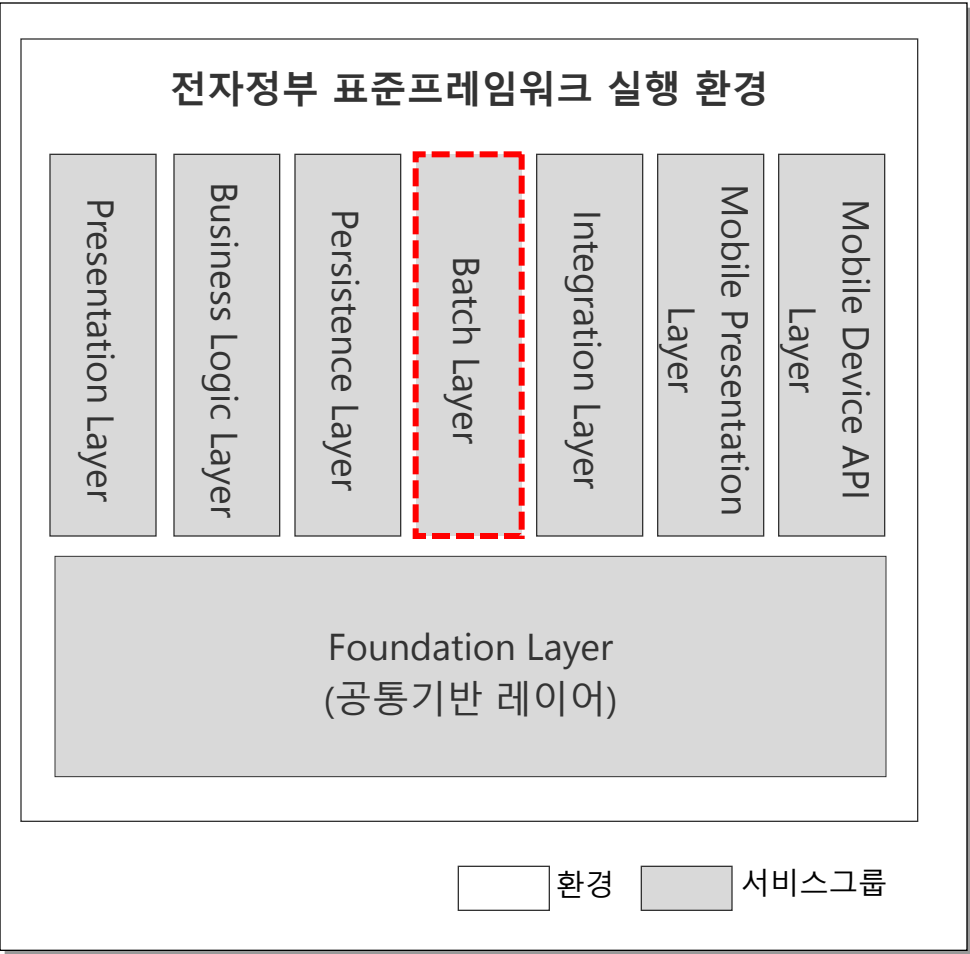
## 5. Batch Processing

## 6. Batch Support

## 7. eGovFrame Batch 제공기능

## 8. 참고자료

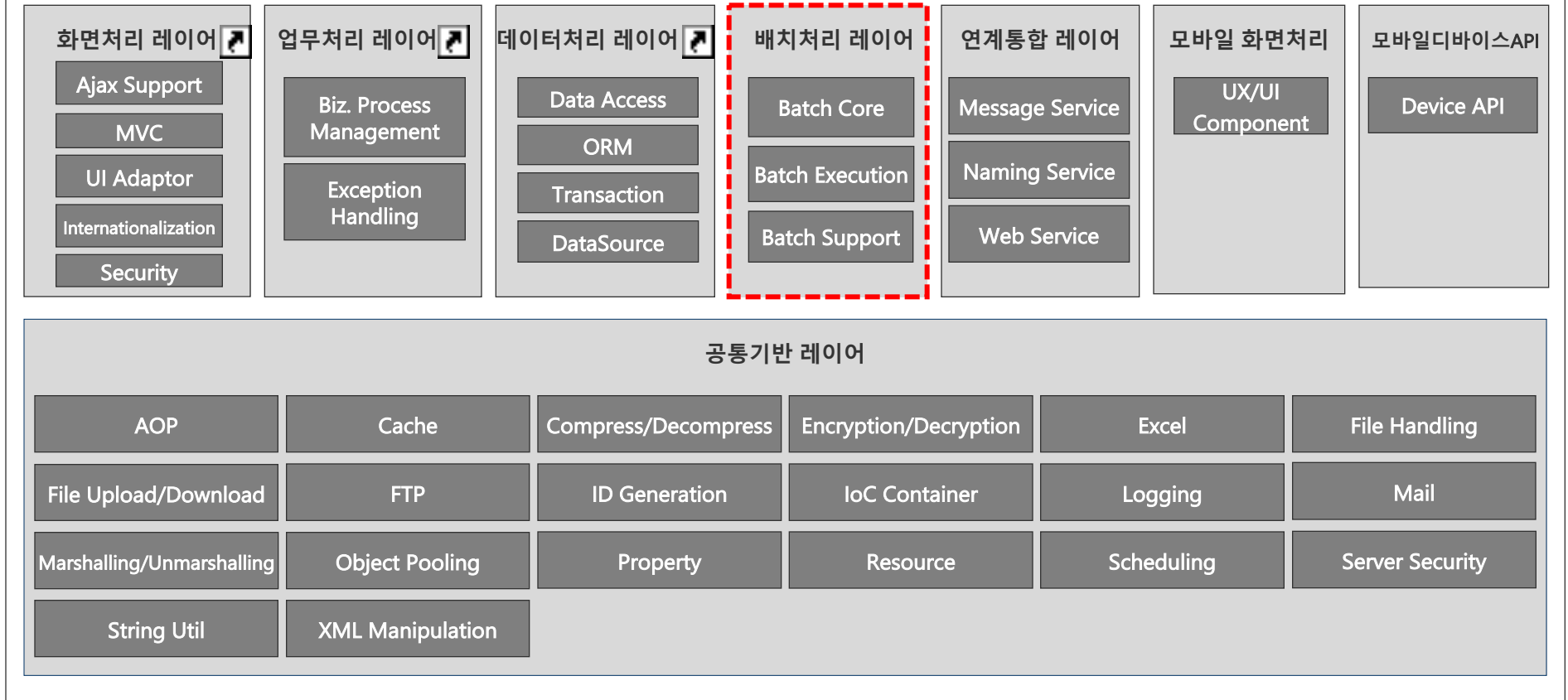
□ 대용량 데이터 처리를 위한 기반 환경을 제공하는 Layer임



서비스 그룹	설명
Presentation Layer	• 업무 프로그램과 사용자 간의 Interface를 담당하는 Layer로서, 사용자 화면 구성, 사용자 입력 정보 검증 등의 기능을 제공함
Business Logic Layer	• 업무 프로그램의 업무 로직을 담당하는 Layer로서, 업무 흐름 제어, 에러 처리 등의 기능을 제공함
Batch Layer	• 대용량 데이터 처리를 위한 기반 환경을 제공하는 Layer임
Persistence Layer	• 데이터베이스에 대한 연결 및 영속성 처리, 선언적인 트랜잭션 관리를 제공하는 Layer임
Integration Layer	• 타 시스템과의 연동 기능을 제공하는 Layer임
Foundation Layer	• 실행 환경의 각 Layer에서 공통적으로 사용하는 공통 기능을 제공함

- 배치처리 레이어는 Batch Core, Batch Execution, Batch Support 등 총 3개의 서비스를 제공함

### 실행 환경



## ❑ Batch 작업의 정의

- 사람의 상호 작용 없이 컴퓨터상에서 일련의 프로그램을 실행시키는 것. [출처:Wikipedia]



- 정기적인 반복 수행, 정해진 규칙에 따라 일괄처리
- 주요 사례 : 대량의 데이터 전송, 웹사이트 로그 통계 분석, 회계 결산, 쇼핑몰 일일 결제 취합 등에 사용

특징	설명
대용량 데이터 처리	- 대량의 데이터를 전송 또는 수신하거나, 계산 처리가 가능해야 함
자동화	- 사용자의 개입없이 특정시간에 자동으로 실행 가능해야 함
견고함	- 유효하지 않은 데이터도 처리할 수 있어야 하며 이로 인해 중단되어서는 안됨
성능	- 지정한 시간 내에 처리를 완료할 수 있어야 함 - 주로 시스템 부하가 적은 새벽시간대에 백그라운드로 실행
신뢰성	- 실행 결과를 확인할 수 있어야 하며, 데이터에 문제발생 시 추적가능해야 함

### ❑ Spring Batch

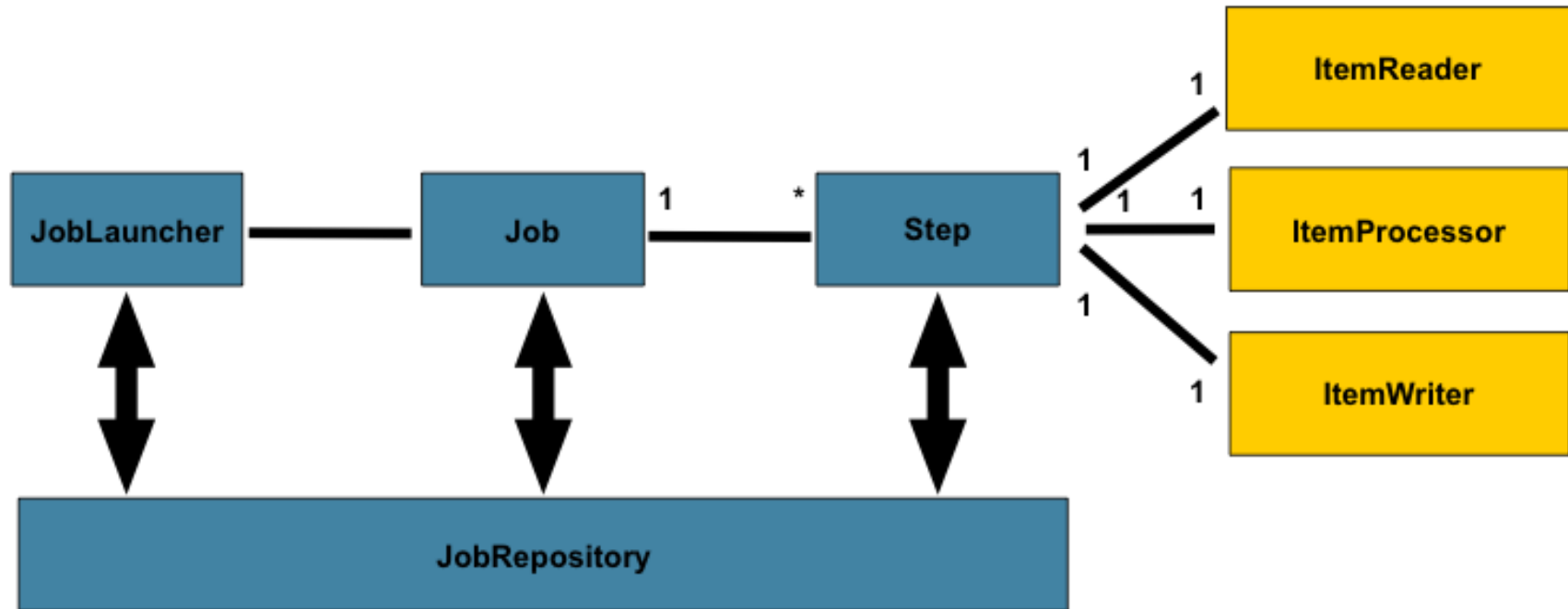
“ **Spring Batch** 는 엔터프라이즈 시스템의 일상적인 업무처리에 필요한 강력한 배치 어플리케이션을 개발할 수 있도록 설계된 가벼우면서, 종합적인 배치 프레임워크 ” [출처:spring.io]

- Spring Source 와 Accenture 사가 협업해서 개발하여 2007년에 탄생
- 배치 영역에서 수많은 경험과 노하우를 가진 Accenture 사의 주도하에 개발
- Spring 의 특성을 그대로 계승하여 DI, AOP, 서비스 추상화 등의 Spring 핵심기능들을 사용 가능

### ❑ Spring Batch 기능

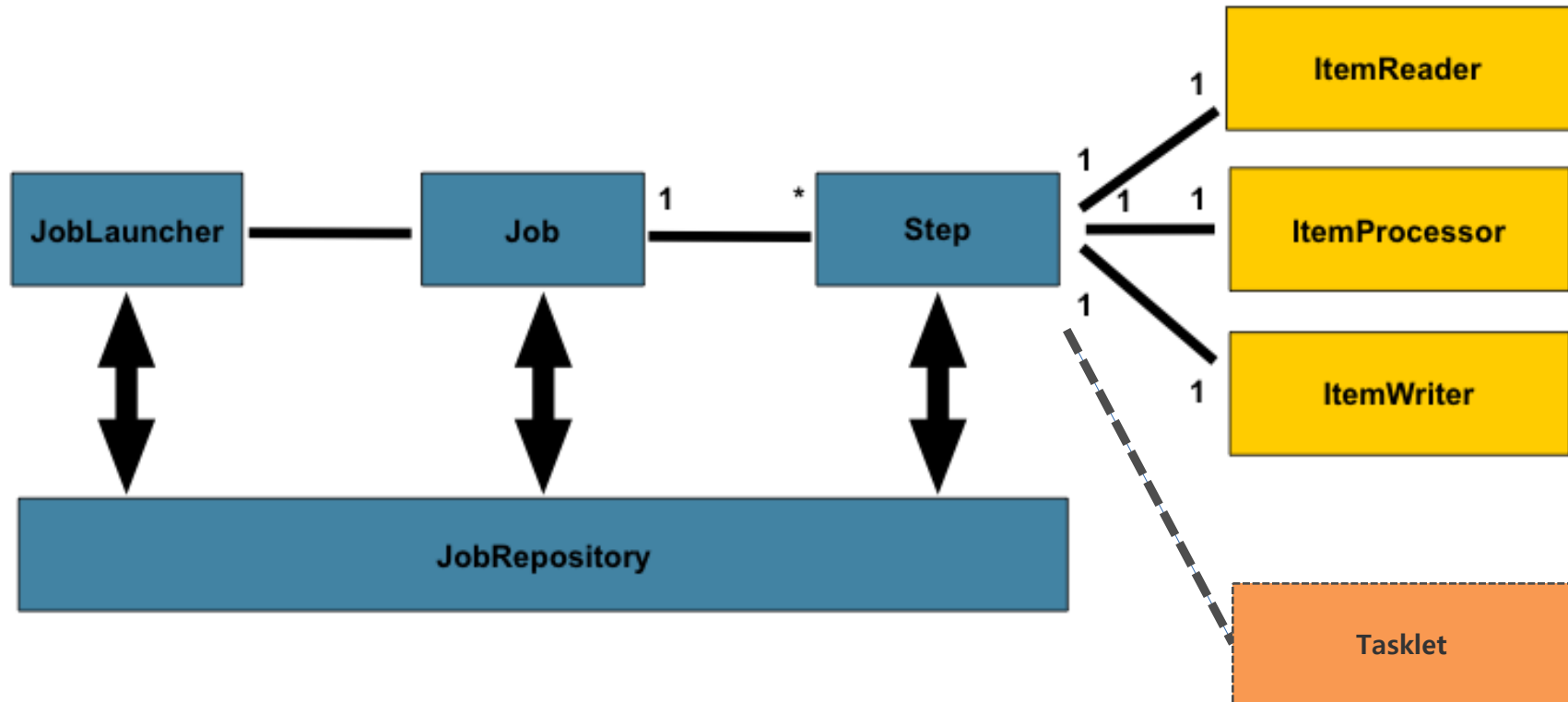
- Batch Monitoring 기능 제공 ( Commit/Rollback/Retry Count )
- Chunk 기반의 데이터 처리 지원 ( Commit Interval )
- Retry, Repeat, Skip 기능 지원
- Quartz, Command Line, Web 등의 외부 트리거를 통한 실행 지원

## □ Spring Batch 기본 구성





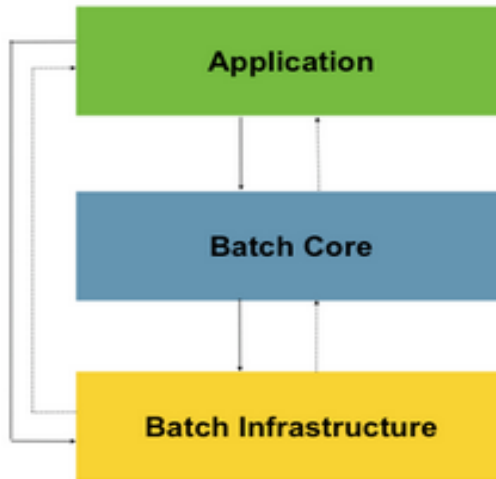
## ❑ Spring Batch 기본 구성



## □ Spring Batch 구성요소

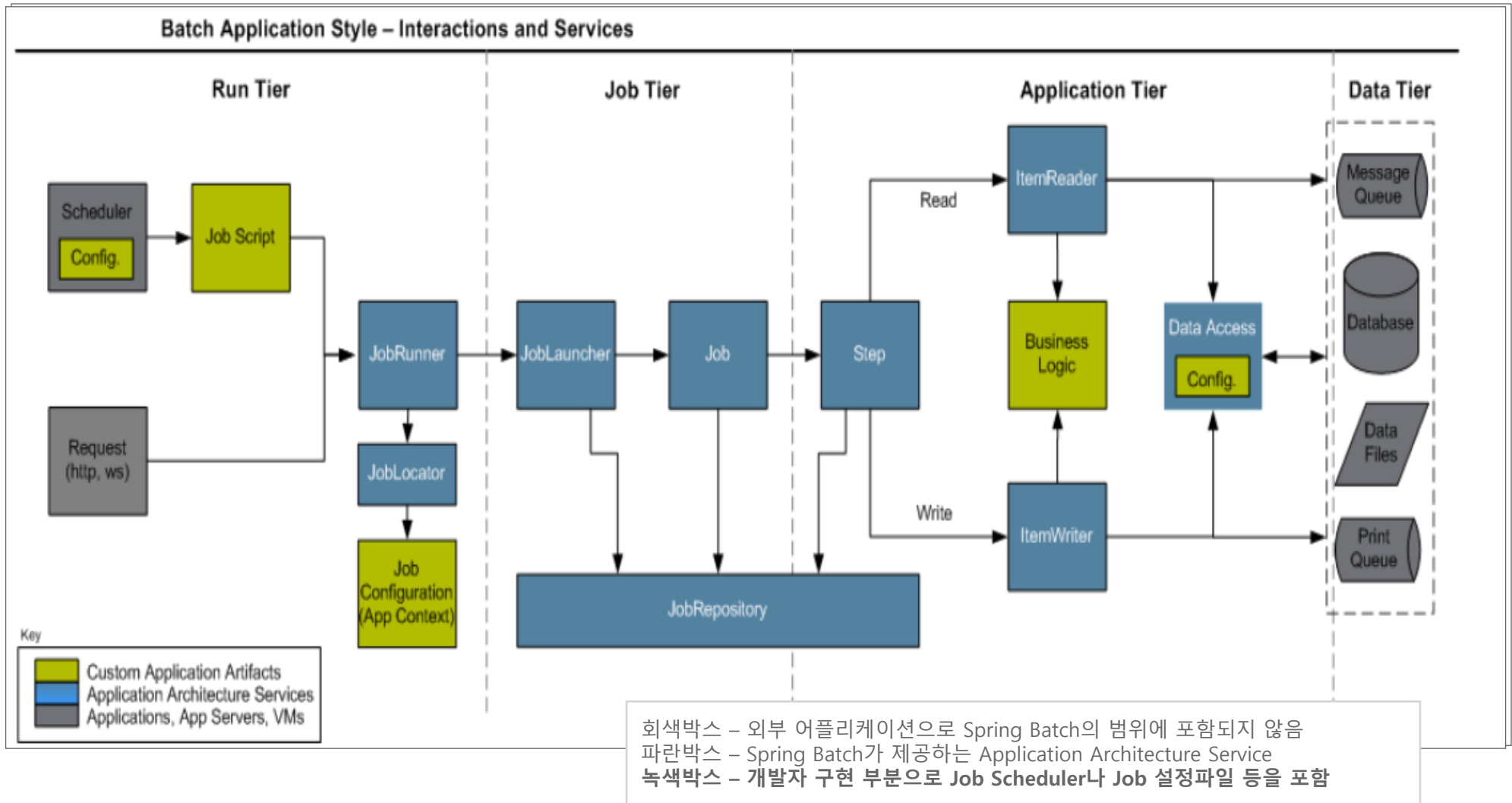
구분	설명
JobLauncher	JobLauncher는 Batch Job을 실행시키는 역할 수행. Job과 Parameter를 받아서 실행하며 JobExecution를 반환.
JobRepository	수행되는 Job에 대한 정보를 담고 있는 저장소. Job이 수행, 종료, 실행 횟수 및 결과 등, Batch수행과 관련된 모든 Meta Data가 저장되어 있음
Job	실행시킬 작업. 논리적인 Job 실행의 개념.
JobParameter	Batch Job을 실행하는데 사용하는 파라미터의 집합으로 Job이 실행되는 동안에 Job을 식별하거나 Job에서 참조하는 데이터로 사용
JobInstance	논리적인 Job 실행 (JobInstance=Job+JobParameter)
Step	Batch Job을 구성하는 독립적인 하나의 단계. Job은 1개 이상의 Step으로 구성 실제 Batch 처리 과정을 정의하고, 제어하는데 필요한 모든 정보를 포함 Step의 내용은 전적으로 개발자의 선택에 따라 구성됨.
Item	처리할 데이터의 가장 작은 구성 요소. (예)파일의 한 줄, DB의 한 Row, Xml의 특정 Element
ItemReader	Step안에서 File 또는 DB등에서 Item을 읽어 들임. 더 이상 읽어올 Item이 없을 때에는 read() 메소드에서 null값을 반환하며 그 전까지는 순차적인 값을 리턴.
ItemWriter	Step안에서 File 또는 DB등으로 Item을 저장.
Item Processor	Item reader에서 읽어 들인 Item에 대하여 필요한 로직처리 작업을 수행.

## ❑ Spring Batch Layered Architecture



서비스 그룹	설명
Application	• Spring Batch를 이용해 개발된 Batch Job과 커스텀 코드, 비즈니스 로직 (개발자에 의해 작성)
Batch Core	• Batch Job을 실행하거나 제어하는데 필요한 설정 또는 구현체 (예: JobLauncher/Job/Step)
Batch Infrastructure	• Application과 Core에서 사용하는 I/O나 기본적인 서비스 기능 (예: Repeat, Retry, Transaction, I/O)

### □ Spring Batch Architecture



## ❑ Spring Batch Architecture

Tier	설명
Run Tier	<ul style="list-style-type: none"><li>• Scheduling과 Application 실행 담당</li><li>• Spring Batch에서는 Scheduling 기능을 따로 제공하지 않고 Quartz같은 외부 모듈이나 Cron을 이용하도록 권고</li></ul>
Job Tier	<ul style="list-style-type: none"><li>• 전체적인 Job 실행 담당</li><li>• Job내의 각 Step들을 지정한 정책에 따라 순차적으로 수행</li></ul>
Application Tier	<ul style="list-style-type: none"><li>• Job을 실행하는데 필요한 컴포넌트들로 구성</li></ul>
Data Tier	<ul style="list-style-type: none"><li>• Database, File 등 물리적 데이터소스와 결합이 이루어지는 영역</li></ul>

### 1. 개요

### 2. Batch 구성요소

- Job Repository
- Job Launcher
- Job Runner
- Job
- Step

### 3. Spring Batch 2.x

### 4. Spring Batch 3.x

### 5. Batch Processing

### 6. Batch Support

### 7. eGovFrame Batch 제공기능

### 8. 참고자료

### ❑ Job Repository 개념

- JobRepository 는 배치 실행중에 발생하는 정보를 저장하는 저장소 역할
- Job의 시작 및 종료시간, 실행횟수, 실행결과 등의 배치 작업과 관련된 메타데이터가 저장됨

### ❑ Job Repository 설정

```
<job-repository id="jobRepository" data-source="dataSource" transaction-manager="transactionManager"
isolation-level-for-create="ISOLATION_SERIALIZABLE" table-prefix="BATCH_" max-varchar-length="1000" />
```

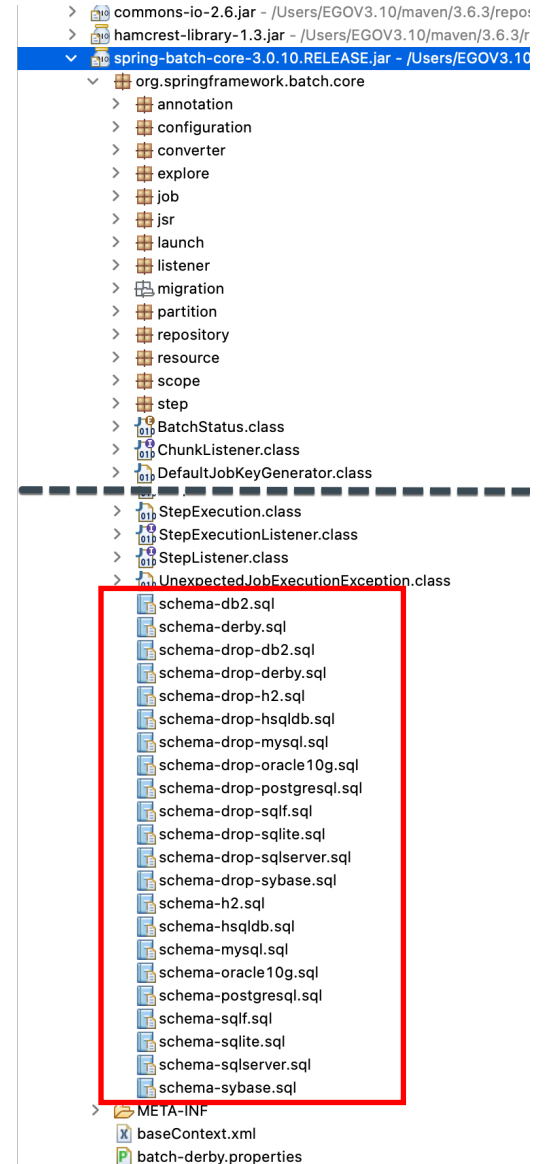
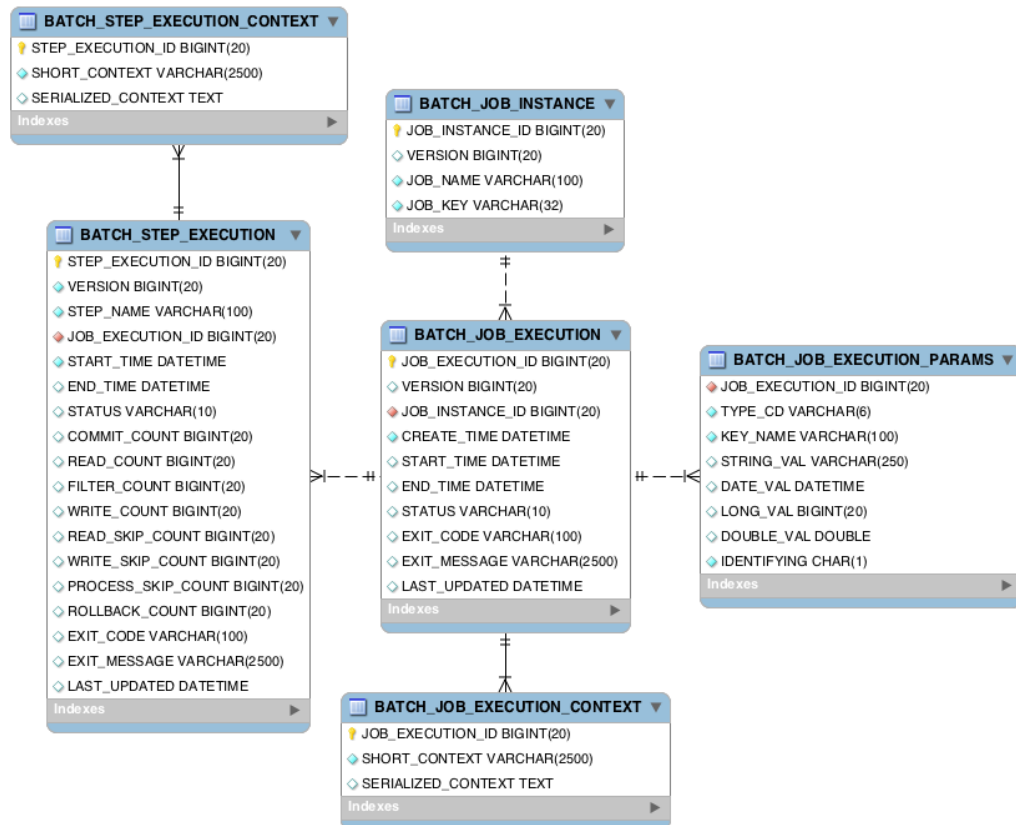
```
<bean id="jobRepository" class="org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="transactionManager" ref="transactionManager" />
  <property name="isolationLevelForCreate" ref="ISOLATION_SERIALIZABLE" />
  <property name="tablePrefix" ref="BATCH_" />
  <property name="maxVarCharLength" ref="1000" />
</bean>
```

### ❑ Database방식과 Memory방식 지원

- DataBase 방식 : Batch 실행 기록을 남겨야 하는 경우, JobRepositoryFactoryBean을 사용 (DataSource 필수)
- Memory 방식 : Batch 실행 기록을 유지하지 않아도 되는 경우, MapJobRepositoryFactoryBean 사용

### ❑ Meta-Data Schema

- Spring Batch 를 DataBase 방식으로 실행하기 위해 반드시 존재해야 함
- Spring Batch Core JAR 파일에 각 DB 종류별 스크립트가 포함되어 있음





## ❑ Job Launcher 개념

- JobLauncher 는 Batch 작업을 실행시키는 역할을 수행.
- Job과 Job Parameters를 이용하여 요청된 Batch 작업을 실행 후 JobExecution을 반환.

## ❑ Job Launcher 설정

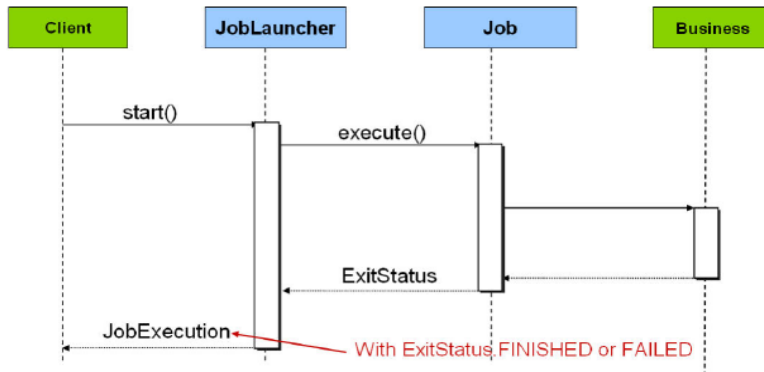
```
public interface JobLauncher {  
    public JobExecution run(Job job, JobParameters jobParameters) throws JobExecutionAlreadyRunningException,  
        JobRestartException, JobInstanceAlreadyCompleteException, JobParametersInvalidException;  
}
```

```
<bean id="jobLauncher"  
    class="org.springframework.batch.core.launch.support.SimpleJobLauncher">  
    <property name="jobRepository" ref="jobRepository" />  
</bean>
```

- JobLauncher Interface의 기본 구현 클래스로는 SimpleJobLauncher가 제공.
- SimpleJobLauncher 클래스는 JobName과 JobParameter를 이용하여 JobRepository에서 JobExecution을 획득하고 작업을 수행함.
- jobRepository 설정은 필수임

## ❑ Job의 동기적 실행과 비동기적 실행

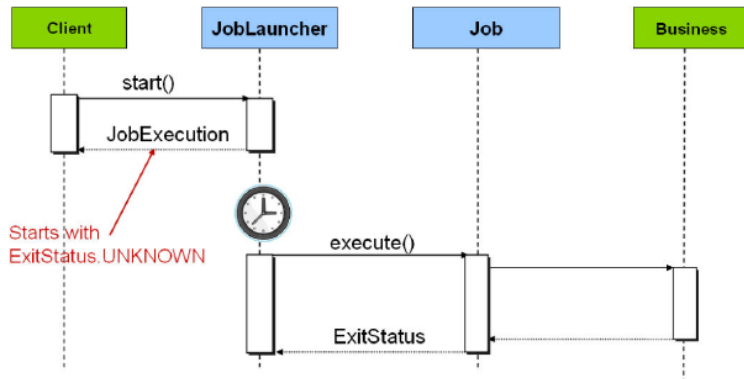
- JobLauncher는 taskExecutor 설정을 통해 Job을 동기적 혹은 비동기적으로 실행 가능.
- 별도로 설정하지 않으면 syncTaskExecutor클래스가 Default로 설정됨.



```

<bean id="jobLauncher"
class="org.springframework.batch.execution.launch.SimpleJobLauncher">
  <property name="jobRepository" ref="jobRepository" />
  <property name="taskExecutor">
    <bean class="org.springframework.core.task.SyncTaskExecutor" />
  </property>
</bean>

```



```

<bean id="jobLauncher"
class="org.springframework.batch.execution.launch.SimpleJobLauncher">
  <property name="jobRepository" ref="jobRepository" />
  <property name="taskExecutor">
    <bean
class="org.springframework.core.task.SimpleAsyncTaskExecutor" />
  </property>
</bean>

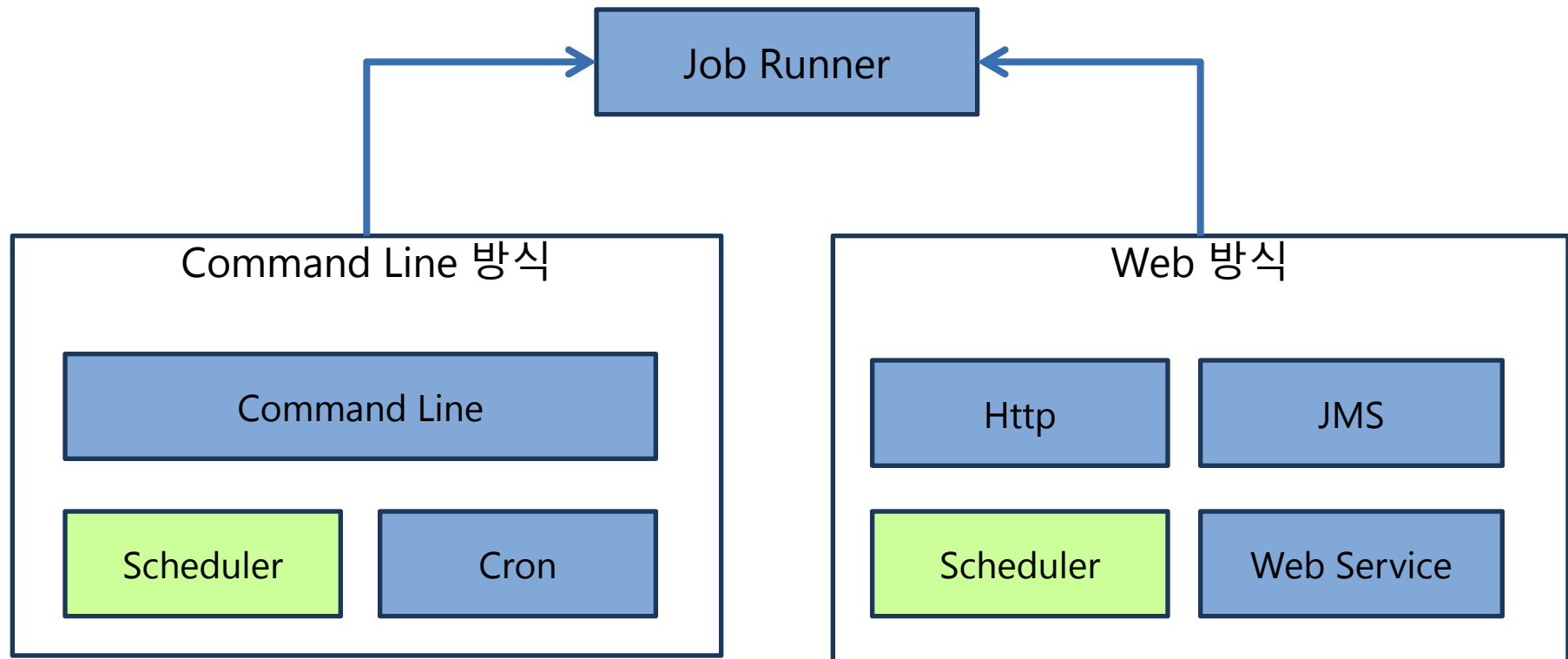
```

## □ 개념

- 외부 실행 모듈과 JobLauncher를 연결해주는 모듈

## □ 유형

- Command Line 방식, Web 방식(Scheduler는 두 가지 방식 모두 가능)



## ❑ Command Line 방식

- Spring Batch에서 제공하는 CommandLineJobRunner라는 클래스를 이용하면 main 메소드가 포함된 일반적인 java application처럼 Spring Batch의 Job들을 실행 가능

```
java CommandLineJobRunner [설정파일명] [job이름] [Job Parameter A=B형태]
```

- 예시

```
bash$ java CommandLineJobRunner endOfDayJob.xml endOfDay schedule.date(date)=2007/05/05
```

## ❑ Web 방식

- Web 방식은 WebApplicationContext에 job Launcher 및 job 설정을 bean으로 등록한 후, Controller 에 Http Request 요청이 왔을 때, Job Launcher의 run() 메소드를 호출하여 작업 수행

```
@RequestMapping(value="/batchRun.do", method = RequestMethod.POST)
public String batchRun(@RequestParam(value = "jobName", required = false) String jobName,
    @RequestParam(value = "async", required = false) String async, Model model){
    try {
        JobExecution jobExecution = jobLauncher.run(jobRegistry.getJob(jobName), getUniqueJobParameters(jobName));
        .....
    }
```

## ❑ Scheduler 설정

- Spring Batch는 Scheduling 기능을 제공하지 않음, Quartz 나 Cron 을 사용하는 것을 권고함
- Command Line과 Web에서 모두 사용 가능

```
<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
  <property name="triggers">
    <bean id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">
      <property name="jobDetail" ref="jobDetail" />
      <property name="cronExpression" value="0/10 * * * * ?" />
    </bean>
  </property>
</bean>

<bean id="jobDetail" class="org.springframework.scheduling.quartz.JobDetailBean">
  <property name="jobClass" value="org.springframework.batch.sample.quartz.JobLauncherDetails" />
  <property name="group" value="quartz-batch" />
  <property name="jobDataAsMap">
    <map>
      <entry key="jobName" value="footballJob"/>
      <entry key="jobLocator" value-ref="jobRegistry"/>
      <entry key="jobLauncher" value-ref="jobLauncher"/>
    </map>
  </property>
</bean>

<bean id="jobRegistry" class="org.springframework.batch.core.configuration.support.MapJobRegistry" />
```

Quartz 를 사용하여 스케줄 설정대로  
Job을 실행하는 예시 코드

## □ Job 개념

- Job은 하나의 배치 실행 단위를 의미 (일감으로 표현 가능)
- 배치처리에 있어 최상위 계층에 있으며, 1개 이상의 Step (단계) 을 포함하는 컨테이너라고 할 수 있음

## □ Job 설정

```
<job id="footballJob" job-repository="jobRepository" restartable="true">
  <step id="playerload" next="gameLoad" />
  <step id="gameLoad" next="playerSummarization" />
  <step id="playerSummarization" />
</job>
```

구분	설명
id	job 식별자
step	job은 적어도 하나 이상의 step을 정의
job-repository	Batch 작업 실행 중 JobExecution을 주기적으로 저장하기 위한 저장소(default 설정은 'jobRepository'로 생략 가능)

## □ Restart 가능 여부 설정

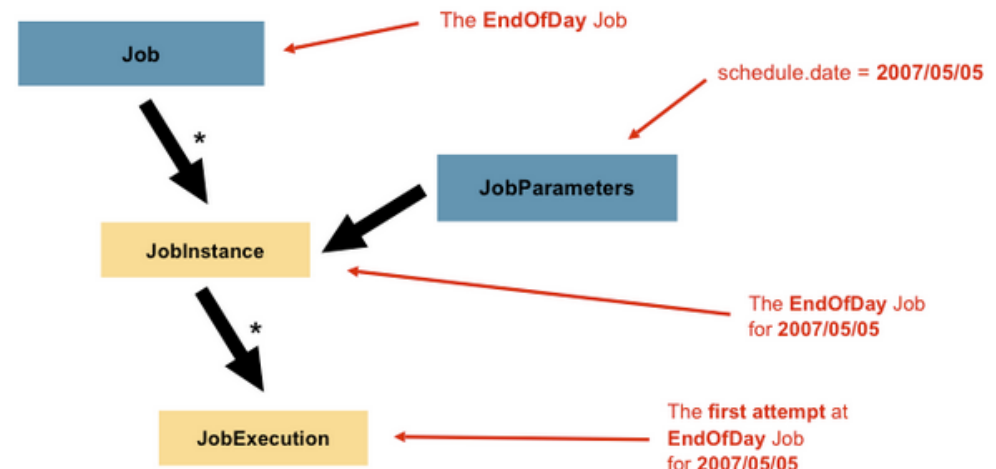
- 동일한 Job Instance에 대해 Job Restart를 가능하게 하는 옵션
- **restartable=false**로 설정된 job을 재시작할 경우 JobRestartException 발생

### ❑ Job Instance

- JobInstance는 논리적인 Job 실행의 개념으로 Job이 실행될 때 생성됨
- JobInstance = Job + JobParameters 로 표현할 수 있음 (고유한 키)
- 배치 실행시에 이미 동일한 JobInstance (Job+JobParameters) 가 존재할 경우, JobInstance는 생성되지 않고 JobExecution 이 생성됨. JobInstance는 1개, JobExecution은 여러개 생성 될 수 있음.

### ❑ Job Parameters

- JobParameters는 하나의 Job에 존재할 수 있는 여러 개의 JobInstance를 구별하기 위한 Parameter 집합임
- Job을 식별하거나 Job에서 참조하는 데이터로 사용됨
- Job Parameter를 통해 여러 JobInstance 생성 가능



## ❑ Job Execution

- JobExecution은 JobInstance에 대해 한번의 Job 실행을 나타내는 객체, 실행 할때마다 생성
- 실행 결과가 COMPLETED 인 JobExecution을 가진 JobInstance는 restart 불가 (allow-start-if-complete 옵션 제공)
- Job에 정의된 Step들을 순서에 따라 실행하고 JobRepository에 실행 정보를 저장.

## ❑ Job Execution Property 정보

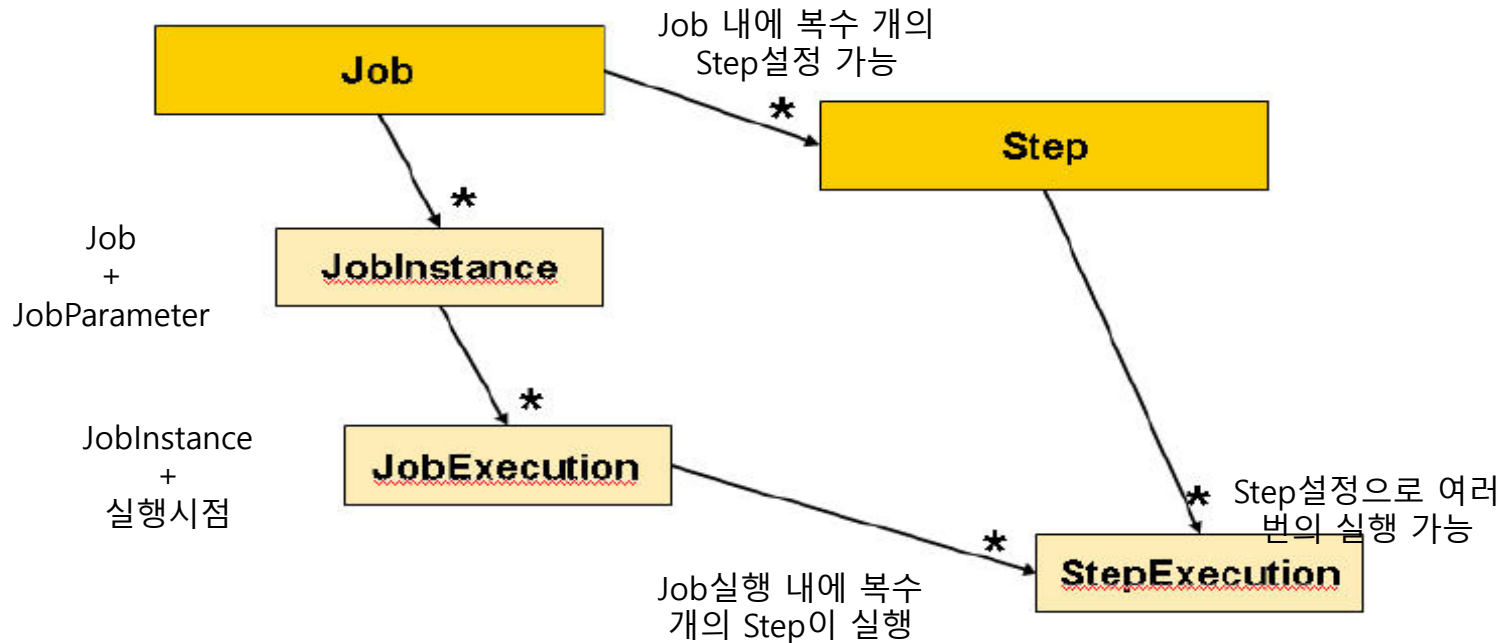
\* 표시 항목은 jobRepository에 저장되는 정보를 나타냄

구분	설명
status *	JobExecution의 상태를 보여주는 지표 - 수행 중: STARTED - 실패: FAILED - 성공: COMPLETED
startTime *	Job이 실행된 시간
endTime *	Job이 종료된 시간 (Job성공 여부와 무관함)
exitStatus *	Job의 실행 결과를 보여주는 상태 값 (COMPLETED, NOOP, FAILED 등)
createTime *	JobExecution 정보가 최초에 생성된 시간 (startTime보다 createTime이 먼저 등록됨)
lastUpdated *	JobExecution 정보가 마지막으로 변경된 시간
executionContext	JobExecution 정보를 담고 있는 공간
failureExceptions	Job수행 동안 발생한 Exception의 List



## □ Step 개념

- Step은 실질적인 배치 처리 내용을 정의하고 있는 객체 (무엇을 어떻게 처리할지에 대한 설정)
- 하나의 Job 에는 최소 1개 이상의 Step(단계) 이 있어야 함
- Step에는 JobExecution에 대응되는 StepExecution이 있음.



## ❑ Step Execution

- StepExecution은 한번의 Step 실행을 나타내는 객체, 매번 시도될 때마다 생성됨.
- StepExecution은 Step이 실행 중에 어떤 일이 일어났는지에 대한 속성들을 저장하는 저장 메커니즘 역할을 하며 commit count, rollback count, start time, end time 등의 Step 상태정보를 저장함.

(commit 시점에 데이터 갱신되며, restart, 통계 용도로 주로 사용됨) ➔ JobRepository로 실행 정보 저장

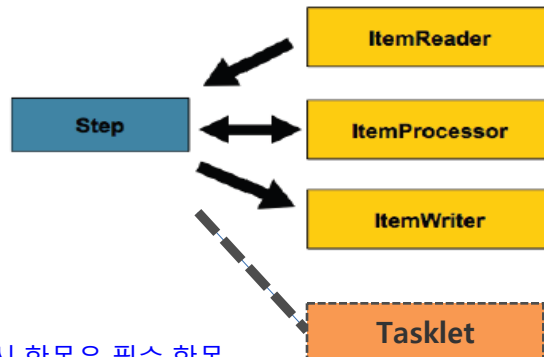
\* 표시 항목은 jobRepository에 저장되는 정보를 나타냄

구분	설명
status *	StepExecution의 상태를 보여주는 지표 ( STARTED/ FAILED/ COMPLETED )
startTime *	Step이 시작된 시간
endTime *	Step이 종료된 시간
exitStatus *	Step의 실행 결과를 보여주는 상태 값
readCount *	성공적으로 읽은 데이터의 개수
writeCount *	성공적으로 쓴 데이터의 개수
commitCount *	StepExecution 동안 Commit된 횟수
rollbackCount *	StepExecution동안 Rollback된 횟수
readSkipCount *	데이터 읽기 도중 실패로 인하여 skip한 횟수
processSkipCount *	데이터 처리 도중 실패로 인하여 skip한 횟수
filterCount *	ItemProcessor에서 Filtering한 데이터의 개수
writeSkipCount *	데이터 쓰기 도중 실패로 인하여 skip한 횟수

## □ Step 설정

– Step 설정 방법은 크게 2가지로 구분할수 있음

1. Chunk지향처리 : 데이터를 item 단위로 읽고 처리하고 쓰는 방식 (Spring Batch 기본 유형)
2. Tasklet 처리 : 단일 태스크나 커스텀한 코드를 수행하기 위한 처리 방법 (Tasklet 인터페이스 직접 구현)



\* 표시 항목은 필수 항목

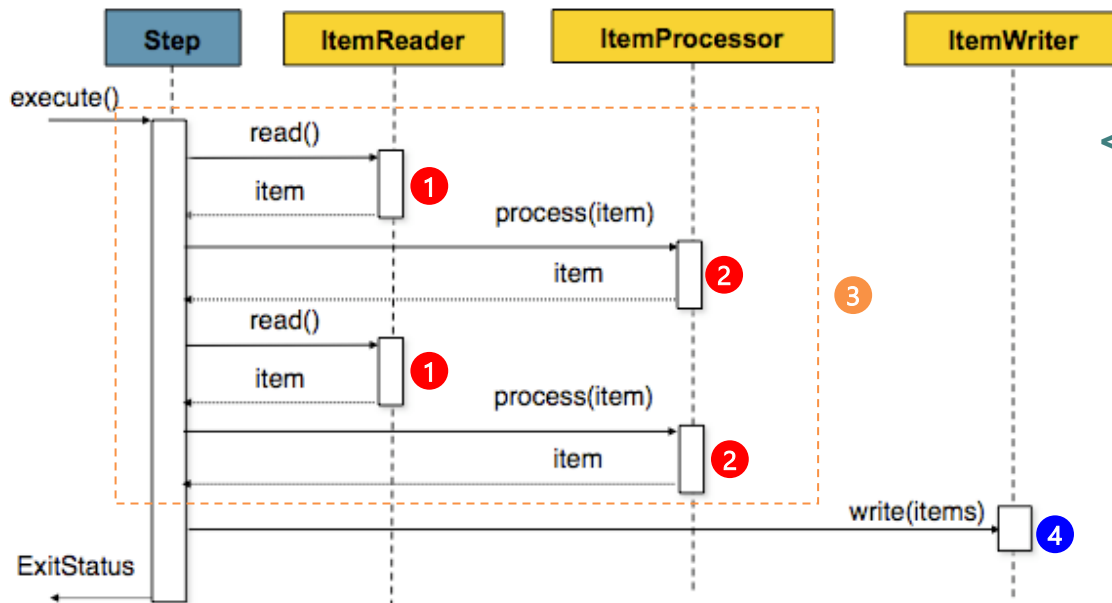
```

<job id="myJob" job-repository="jobRepository">
  <step id="myStep">
    <tasklet transaction-manager="transactionManager">
      <chunk
        reader="itemReader"
        processor="itemProcessor"
        writer="itemWriter"
        commit-interval="10" />
      </tasklet>
    </step>
  </job>
  
```

구분	설명
reader *	item 단위로 데이터를 읽어들이는 역할
processor	reader가 읽어들이 데이터를 받아서 처리(가공)하는 역할 (processor는 생략가능, 생략할 경우 reader가 읽어들이 데이터를 writer로 바로 전달함)
writer *	reader가 읽어들이 데이터나 processor가 처리한 데이터를 저장하는 역할
transaction-manager *	Spring의 PlatformTransactionManager로 Batch작업 중 트랜잭션을 시작하고 커밋하는데 사용 (default 설정은 "transactionManger"이며 생략 가능)
commit-interval *	트랜잭션내에서 처리할 데이터의 묶음 개수 (chunk 사이즈)

### ❑ Chunk 지향 처리 (Chunk-Oriented Processing)

- Spring Batch에서 가장 일반적으로 사용하는 Step 유형
- 데이터를 일정크기의 Chunk(덩어리) 단위로 데이터를 처리하는 방법
- 읽기(Read) → 처리(Processor) → 쓰기(Write)의 단계를 거치는 메커니즘
- 처리할 데이터가 없을때까지 반복해서 실행

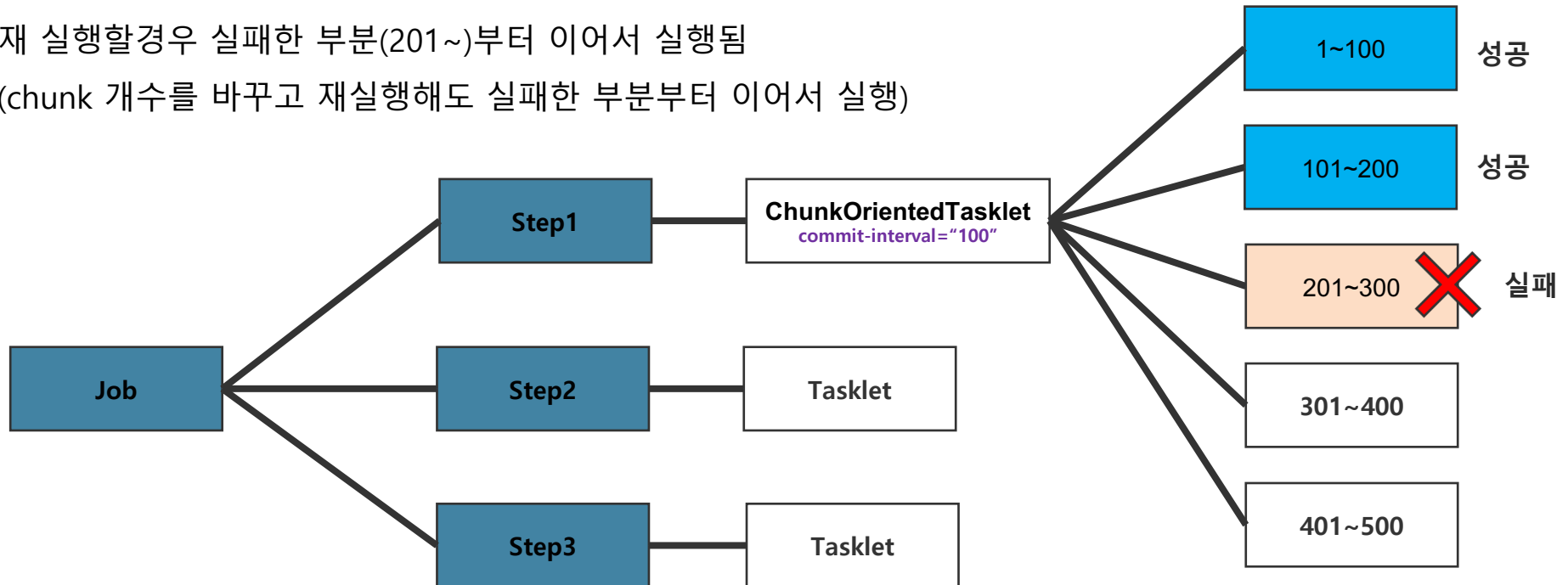


#### <chunk commit-interval="2" /> 인 경우 예시

- 1 Reader가 1개의 데이터를 읽어옴
- 2 Processor가 1개의 데이터를 처리함
- 3 지정된 Chunk 단위만큼 쌓일때까지 위 1, 2단계를 반복 수행
- 4 지정된 Chunk 단위만큼 쌓이면 Writer에 일괄 전달하여 저장함

### ❑ Chunk 지향 처리 – 트랜잭션 (예시)

- Step1의 Chunk 속성값으로 commit-interval="100" 지정 후 배치 실행했을 경우
- 250번째 데이터를 읽고 처리하는 도중 오류가 발생했을 경우
- 1~200번 데이터까지만 저장됨
- 201~249번까지 읽고 처리한 데이터는 저장되지 않음
- 재 실행할 경우 실패한 부분(201~)부터 이어서 실행됨  
(chunk 개수를 바꾸고 재실행해도 실패한 부분부터 이어서 실행)



## ❑ Tasklet 인터페이스 직접 구현

- 단일 태스크 또는 커스텀한 코드를 수행하기 위한 처리 방법 (유저 커스텀)
- 예를 들어 단순히 파일의 복사 및 이동, DB의 프로시저 호출등의 단순한 처리를 메소드 하나로 구현하고 싶을 경우 사용
- JSR-352의 Batchlet 에 대응하는 인터페이스 (chunk 지향처리 이외의 모든 처리 유형)

```
<job id="myJob">
  <step id="myStep">
    <tasklet ref="myTasklet" />
  </step>
</job>
<bean id="myTasklet" class="egovframework.example.bat.tasklet.MyTasklet" ></bean>
```

```
public class MyTasklet implements Tasklet {

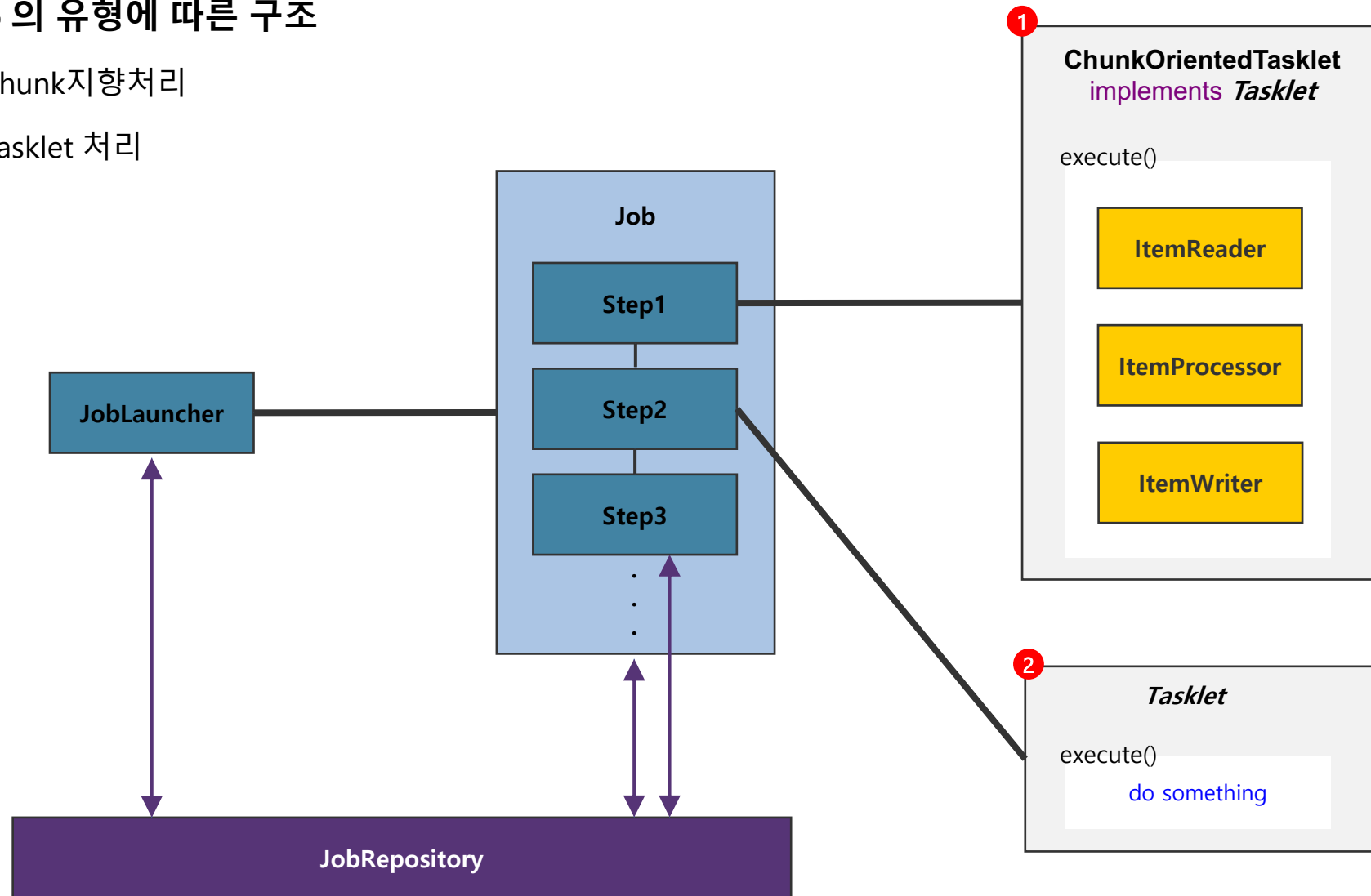
    @Override
    public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) throws Exception {

        System.out.println("do something...");

        return RepeatStatus.FINISHED;
    }
}
```

### □ Step 의 유형에 따른 구조

1. Chunk지향처리
2. Tasklet 처리



### ❑ start-limit

- Step 실행 횟수를 제한하는 옵션 ( default : Integer.MAX\_VALUE )

```
<step id="step1">
  <tasklet start-limit="1">
    <chunk reader="itemReader" writer="itemWriter" commit-interval="10" />
  </tasklet>
</step>
```

### ❑ allow-start-if-complete

- Job을 Restart 할 경우 "COMPLETED"로 완료된 Step의 실행 여부를 설정하는 옵션
- true로 설정 시, "COMPLETED"로 완료한 Step도 다시 실행됨
- false 설정 시, "COMPLETED"로 완료한 Step은 Skip 됨
- (아래 예시)

"step1" Step은 10번만 실행 가능하며, Job을 Restart 했을 경우 이전 실행결과에 관계없이 재실행됨

```
<step id="step1">
  <tasklet start-limit="10" allow-start-if-complete="true">
    <chunk reader="itemReader" writer="itemWriter" commit-interval="20" />
  </tasklet>
</step>
```



### ❑ Retry

- Batch작업 수행 중 오류 발생 시 지정한 횟수만큼 반복해서 처리하는 옵션
- Batch수행 중 Exception발생 시 3번만큼 재시도 하는 예제
- org.springframework.retry 패키지를 디버깅해보면 retry 동작 확인 가능

```
<job id="retryJob">
  <step id="step1">
    <tasklet>
      <chunk
        reader="itemGenerator"
        writer="itemWriter"
        commit-interval="10"
        retry-limit="3">

        <retryable-exception-classes>
          <include class="java.lang.Exception" />
        </retryable-exception-classes>

      </chunk>
    </tasklet>
  </step>
</job>
```

## □ Skip

- 데이터를 처리하는 도중 Exception이 발생했을 때, 해당 데이터 처리를 건너뛰는 옵션
- (아래 예시) 데이터 처리 도중 java.lang.RuntimeException을 10회까지 허용하고, java.io.FileNotFoundException은 Skip 대상에서 제외하는 설정

```
<job id="skipJob">
  <step id="step1" parent="baseStep">
    <tasklet>
      <chunk
        reader="fileItemReader"
        processor="tradeProcessor"
        writer="tradeWriter"
        commit-interval="3"
        skip-limit="10">

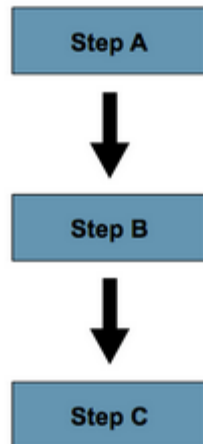
        <skippable-exception-classes>
          <include class="java.lang.RuntimeException" />
          <exclude class="java.io.FileNotFoundException" />
        </skippable-exception-classes>

      </chunk>
    </tasklet>
  </step>
</job>
```

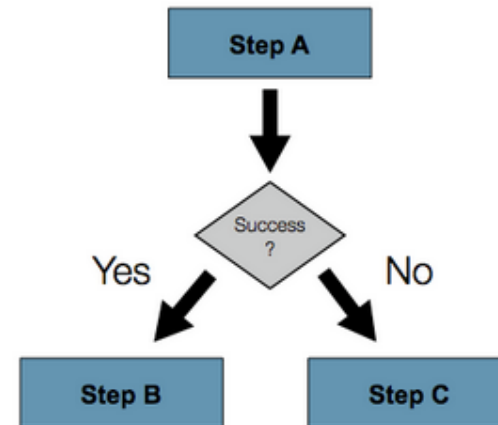
### □ Step Flow Control

- Step 내의 next 설정과 Decision 설정으로 Job을 수행할 수 있음
- Step의 처리결과에 따라 다른 Step을 선택하여 수행할 수 있고, 특정 Step의 실패가 Job 전체의 실패로 이어지지 않도록 구성할 수 있음

[순차적인 Step 실행]



[조건에 따른 Step 실행]



**설명**

- 모든 Step을 순서대로 실행
- Step 엘리먼트의 'next' 어트리뷰트를 이용해서 설정
- Spring Batch의 XML 설정은 Job 설정 최상단 Step이 최초로 실행되며, 그 후의 Step 실행 순서는 XML 설정 순서와는 관계 없음

**소스 코드**

```

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

**설명**

- 조건에 따라 Step 실행
- 'next' 엘리먼트는 사용 횟수에 제한이 없으며, 실행 실패에 대한 default 설정이 없음

**소스 코드**

```

<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>
    
```

## □ Batch Status와 Exit Status의 관계

구분	Batch Status	Exit Status
특징	Job과 Step의 수행 상태 표현	Step의 수행이 완료된 시점의 상태 표현
상태 값	COMPLETED, STARTING, STARTED, STOPPING, STOPPED, FAILED, ABANDONED or UNKNOWN	EXECUTING, COMPLETED, NOOP, STOPPED, FAILED, UNKNOWN & Custom Exit(사용자 정의 가능)
활용	배치 운영환경에서 Job의 상태를 확인하고자 할 경우에 사용	Step이나 decision의 next on에 해당하는 조건값으로 사용
표현	JobExecution, StepExecution 테이블의 status항목에 기록	JobExecution과 StepExecution 테이블의 ExitCode 항목에 기록

### ❑ Item 종류

- Item의 종류로는 대표적으로 **FlatFile, XML, Database** 등이 있음
- Spring Batch에서는 많이 사용되는 Item 유형의 Reader와 Writer 구현체를 제공
- 개발자가 직접 Reader, Writer 인터페이스를 구현하여 개발 가능
- ItemProcessor는 별도로 구현해야함

### ❑ ItemProcessor

- ItemProcessor는 아이템 변환 및 처리를 목적으로 하고, Generic 개념의 도입으로 타입 안정성까지 강화

```
public interface ItemProcessor<I, O> {  
    process(I item) throws Exception;  
}
```

- Foo타입의 input을 받아 Bar타입으로 변환하는 예

```
public class FooProcessor implements ItemProcessor<Foo,Bar>{  
    public Bar process(Foo foo) throws Exception {  
        //Perform simple transformation, convert a Foo to a Bar  
        return new Bar(foo);  
    }  
}
```

## ❑ ItemReader 의 종류

- Flat File, XML, Database 등의 데이터 타입을 입력 받을 수 있음

ItemReader	설명
AggregatItemReader	DelegatingItemReader를 확장한 클래스. Read 메소드의 결과로 java.util.Collection객체를 반환함. 직접 파일을 READ하지 않고 대신 READ할 클래스를 지정함(주로 Parallel 작업에 사용됨)
FlatFileItemReader	플랫 파일을 읽어와서 read() 메소드에서 String을 반환함. Flat file에서 item을 읽어들이며 ItemStream을 구현하고 있음.
StaxEventItemReader	StAX를 통해서 XML파일에서 Item을 읽음
JdbcCursorItemReader	JDBC를 이용해서 DB에서 Item을 읽어옴
HibernateCursorItemReader	하이버네이트의 HQL을 사용해서 커서 기반으로 Item조회를 함.
IbatisPagingItemReader	iBatis를 통해 Driving query 기반으로 Item을 읽어들임
JmsItemReader	read메서드에서 javax.jms.Message객체를 반환. Spring의 JmsOperations 객체의 receive메소드를 통해서 Item을 조회함
JpaPagingItemReader	JPQL 문에 기반하여 row를 페이지 단위로 읽어 큰 데이터를 읽을 때 메모리 부족이 생기지 않음
JdbcPagingItemReader	SQL 문에 기반하여 row를 페이지 단위로 읽어 큰 데이터를 읽을 때 메모리 부족이 생기지 않음

## ❑ ItemWriter의 종류

- Flat File, XML, Database 등을 지원함

ItemWriter	설명
AbstractItemStreamItemWriter	ItemStream and ItemWriter interfaces를 결합하여 만든 Abstract Class 가장 기본적인 ItemWriter Interface
CompositemItemWriter	Item을 List로 담겨진 여러 개의 ItemWriter에게 전달하여 처리한다
ItemWriterAdapter	다른 어플리케이션의 클래스를 Spring Batch에 적용할 수 있도록 하는 ItemWriter interface이다.
PropertyExtractingDelegatingItemWriter	ItemWriter 인터페이스를 구현하지 않은 기존의 클래스를 ItemWriter로 쓰고자 할 때, 이 ItemWriter의 구현체를 써서 기존 클래스에 파라미터로 넘겨질 속성값들을 지정하고 실행할 수 있다
FlatFileItemWriter	Item을 String으로 변환해서 Flat File에 쓸 수 있게 하는 ItemWriter
HibernateItemWriter	Hibernate를 이용하여 DB로 Item 을 쓸 수 있게 하는 ItemWriter
JdbcBatchItemWriter	Jdbc를 이용해 Batch Update의 형태로 DB로 Item 을 쓸 수 있게 하는 ItemWriter
JpaItemWriter	JPA EntityManager 인식 및 핸들링할 수 있는 ItemWriter
StaxEventWriterItemWriter	StAX를 이용해서 Item을 XML파일에 쓰는 ItemWriter

1. 개요
2. Batch 구성요소
3. Spring Batch 2.x
4. Spring Batch 3.x
5. Batch Processing
6. Batch Support
7. eGovFrame Batch 제공기능
8. 참고자료



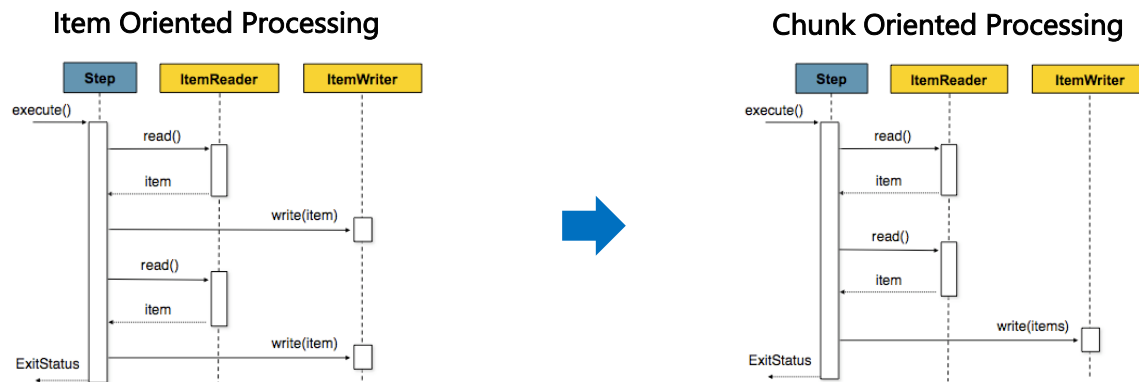
## ❑ Java 5

- Annotation, Generic 지원
- Spring Batch 지원 Annotation

@BeforeJob	@AfterJob	@BeforeStep	@AfterStep
@BeforeRead	@AfterRead	@OnReadError	@OnProcessError
@BeforeProcess	@AfterProcess	@OnWriteError	@OnSkipInRead
@BeforeWrite	@AfterWrite	@OnSkipInProcess	@OnSkipInWrite

## ❑ Chunk Oriented Processing

- 1.x에서는 Item 기반으로 작업 처리
- Chunk 기반 처리는 지정된 크기만큼 Item을 읽고 처리한 후 이 결과를 List의 형태로 가지고 있다가 한번에 쓰는 방식



## ❑ Xml Namespace

- 1.x 에서는 XML 설정 시 모든 설정을 Bean으로 등록
- 2.0 에서는 XML Namespace가 추가되어 XML 설정 상에서 Job과 Step과의 관계를 명시적으로 확인할 수 있으며 사용하고 있는 Item Reader/writer 파악 용이

### [ Spring Batch 1.x ]

```
<bean id="basicJob" parent="simpleJob">
  <property name="steps">
    <bean id="Step1" parent="simpleStep">
      <property name="startLimit" value="300" />
      <property name="commitInterval" value="1000" />
      <property name="streams" ref="itemReader2" />
      <property name="itemReader"/>
      <property name="itemWriter"/>
    </bean>
  </property>
</bean>
```

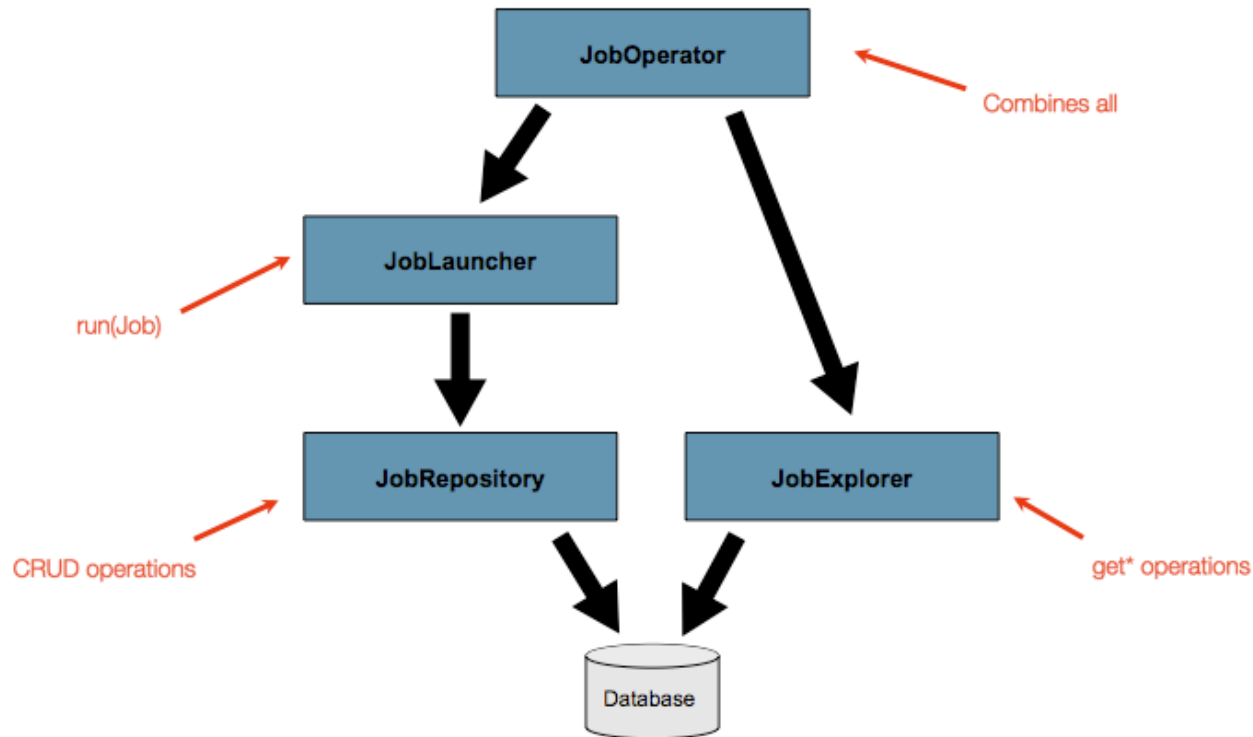


### [ Spring Batch 2.x ]

```
<job id="basic_Job" xmlns="http://www.springframework.org/schema/batch">
  <step id="Steps1" next="ne">
    <tasklet>
      <chunk reader="reader" processor="processor" writer="addr_writer"
        commit-interval="1000"/>
    </tasklet>
  </step>
</job>
```

## ❑ Meta Data enhancements

- 1.x에서는 Batch 작업 실행 중인 상태에서 JobRepository에 직접 접근 불가
- 2.x에서는 JobExplorer와 JobOperator가 추가되었고 이를 통해 JobRepository에 접근할 수 있게 되어 Batch작업 수행 도중 Meta Data를 조회하거나 실행 중인 Job을 제어할 수 있음

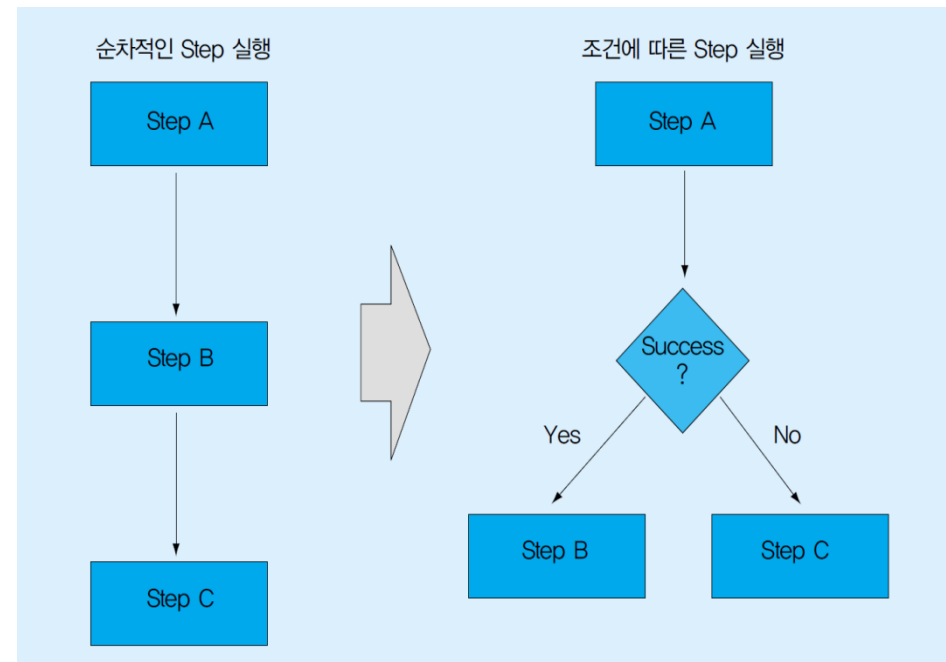


## ❑ Non Sequential Step Execution

- 1.x의 경우 모든 Step의 실행이 순차적으로 이루어져 예외 상황 발생시 해당 Item은 Skip하고 다음 Item을 처리하거나 해당 Job의 Fail 처리를 Listener나 ItemProcessor를 통해 제어
- 2.0은 추가된 Control Flow를 이용하면 현재 진행 중인 작업의 처리 결과 상태(Exit Status)에 따라 수행할 다음 Step을 지정하거나 우회된 Step으로 진행하거나 Job 중지가 가능

조건처리	Property	설명
Next	on – 처리 결과 상태(ExitStatus) to – 다음 실행될 Step ID	on에 지정된 처리 결과 상태가 발생하는 경우 to에 지정된 Step을 실행
Fail	on – 처리 결과 상태 exit-code – JobRepository에 저장 반환할 exitStatus 명	on에 지정된 처리 결과 상태가 발생하는 경우 실패로 간주하고 exit-code에 설정한 처리 결과 상태를 반환한다.
Stop	on – 처리 결과 상태 restart – 재시작 시 수행할 Step ID	on에 지정된 처리 결과 상태가 발생하는 경우 Job의 실행을 멈추고 해당 Job이 다시 실행하는 경우에 restart에 설정한 Step부터 다시 시작한다.
End	on – 처리 결과 상태	on에 지정한 처리 결과 상태가 발생하면 해당 Job 종료로 간주한다.

[조건처리]



[ Step 제어 방식의 변화 ]

1. 개요
2. Batch 구성요소
3. Spring Batch 2.x
4. Spring Batch 3.x
5. Batch Processing
6. Batch Support
7. eGovFrame Batch 제공기능
8. 참고자료

### ❑ Java 7

- Java String Switch
- Multi-Exception Catch
- NIO 2, Diamond Operator 지원, 등 ...

### ❑ Java 8

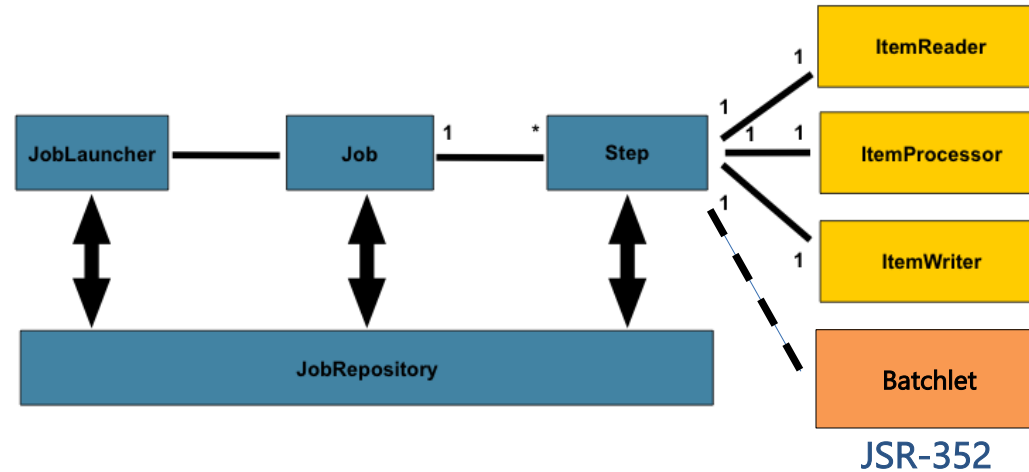
- Lambda Expression
- IO/NIO 확장
- Joda Date/Time API (JSR 310) 지원, 등 ...

### ❑ Spring Batch Integration

- Spring Batch Admin 하위 모듈에서 Spring Batch 독립 모듈로 승격 되면서 **Spring Framework 4 지원**
- messages를 통한 jobs run 기능
- information messages 대한 feedback 기능 제공
- 비동기 ItemProcessors 지원

## ❑ JSR-352 Support

- 자바 표준 배치 스펙 (JSR-352)을 지원하여 쉽고 편리한 범용적인 배치 작업 제공



## ❑ JobScope Support

- Bean Scope의 Job Life Cycle 지원

## ❑ SQLite Support

- JobRepository에서 SQLite 사용할 수 있도록 지원

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.sqlite.JDBC" />
  <property name="url" value="jdbc:sqlite:repository.sqlite" />
  <property name="username" value="" />
  <property name="password" value="" />
</bean>
...
<bean id="jobRepository" class="org.springframework.batch.core.repository.support.JobRepositoryFactoryBean"
  p:dataSource-ref="dataSource" p:transactionManager-ref="transactionManager"
  p:lobHandler-ref="lobHandler"
  p:databaseType="sqlite" />
```

1. 개요
2. Batch 구성요소
3. Spring Batch 2.x
4. Spring Batch 3.x
5. Batch Processing
  - Flat File 처리
  - XML File 처리
  - Multi File 처리
  - DB 처리
  - 기존 서비스 재사용
6. Batch Support
7. eGovFrame Batch 제공기능
8. 참고자료



## ❑ Flat File

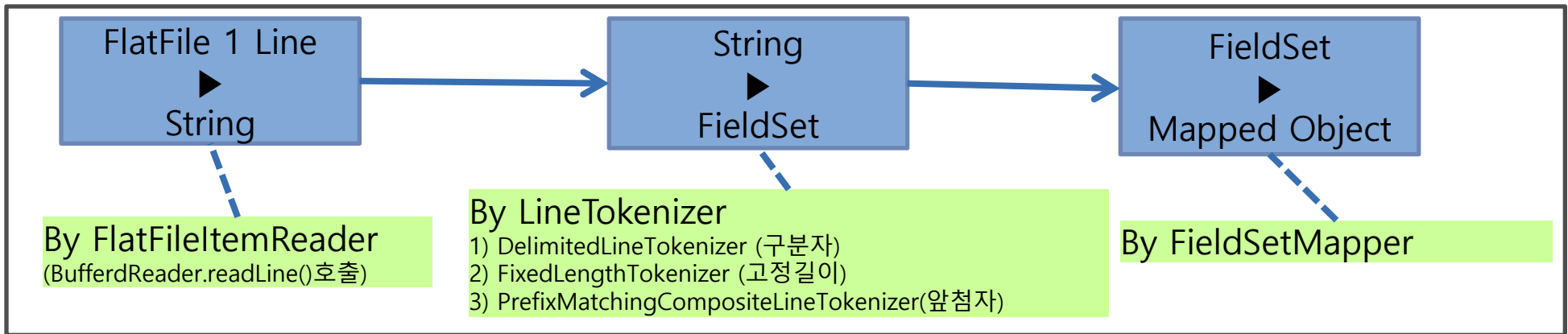
- 플랫폼파일은 2차원 데이터를 포함하는 유형의 파일로 Flat File을 읽고 파싱하는 기본적인 기능을 제공함.

## ❑ Flat File의 두 가지 Token 방식

구분	설명
구분자 (Delimited) 방식	구분자(delimiter)를 이용하여 Element를 구분함
	<ol style="list-style-type: none"> <li>1. 파일에서 한 줄 읽기</li> <li>2. 한 줄 문자열을 LineTokenizer.tokenize() 메소드에 전달해서 FieldSet을 받아옴</li> <li>3. 반환 받은 FieldSet을 FieldSetMapper에게 전달하고, 그 결과 객체를 ItemReader.read()에서 반환.</li> </ol> <pre> FlatFileItemReader itemReader = new FlatFileItemReader(); itemReader.setResource(new FileSystemResource("resources/players.csv")); //DelimitedLineTokenizer defaults to comma as it's delimiter itemReader.setLineTokenizer(new DelimitedLineTokenizer()); itemReader.setFieldSetMapper(new PlayerFieldSetMapper()); itemReader.open(new ExecutionContext()); Player player = (Player)itemReader.read(); </pre>
고정길이 (Fixed Length) 방식	전문과 같이 고정길이의 문자열을 통해 Element를 구분함
	<ol style="list-style-type: none"> <li>1. 파일에서 한 줄 읽기</li> <li>2. FixedLengthLineTokenizer() 메소드 이용 FieldSet을 받아옴</li> <li>3. 반환 받은 FieldSet을 FieldSetMapper에게 전달하고, 그 결과 객체를 ItemReader.read()에서 반환.</li> </ol> <pre> &lt;bean id="fixedLengthLineTokenizer" class="org.springframework.batch.io.file.transform.FixedLengthTokenizer"&gt; &lt;property name="names" value="ISIN, Quantity, Price, Customer" /&gt; &lt;property name="columns" value="1-12, 13-15, 16-20, 21-29" /&gt; &lt;/bean&gt; </pre>

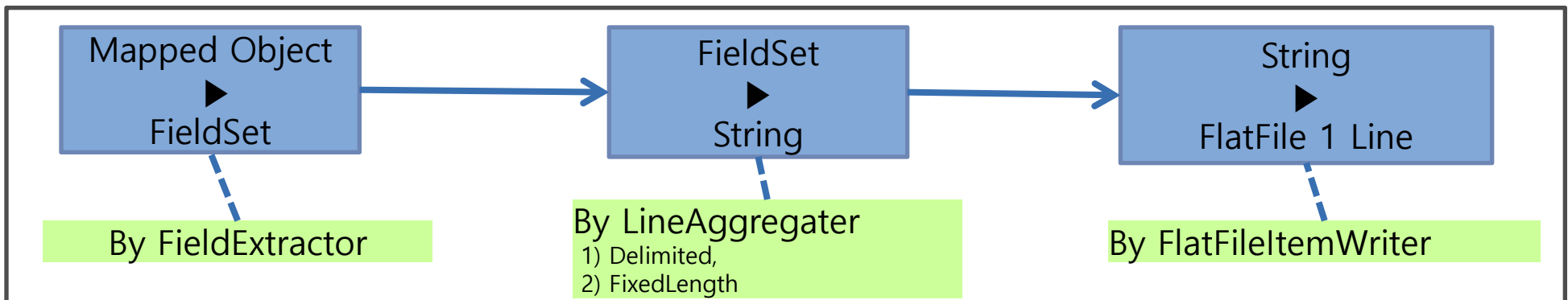
### Flat File Read Mechanism

- Item을 한 라인 씩 읽어서 String으로 변환 후에 VO로 변환



### Flat File Write Mechanism

- Object List형태의 Item을 받아와서 StringBuffer에 한 라인씩 Add한 후 FileWriter를 통해 File로 변환

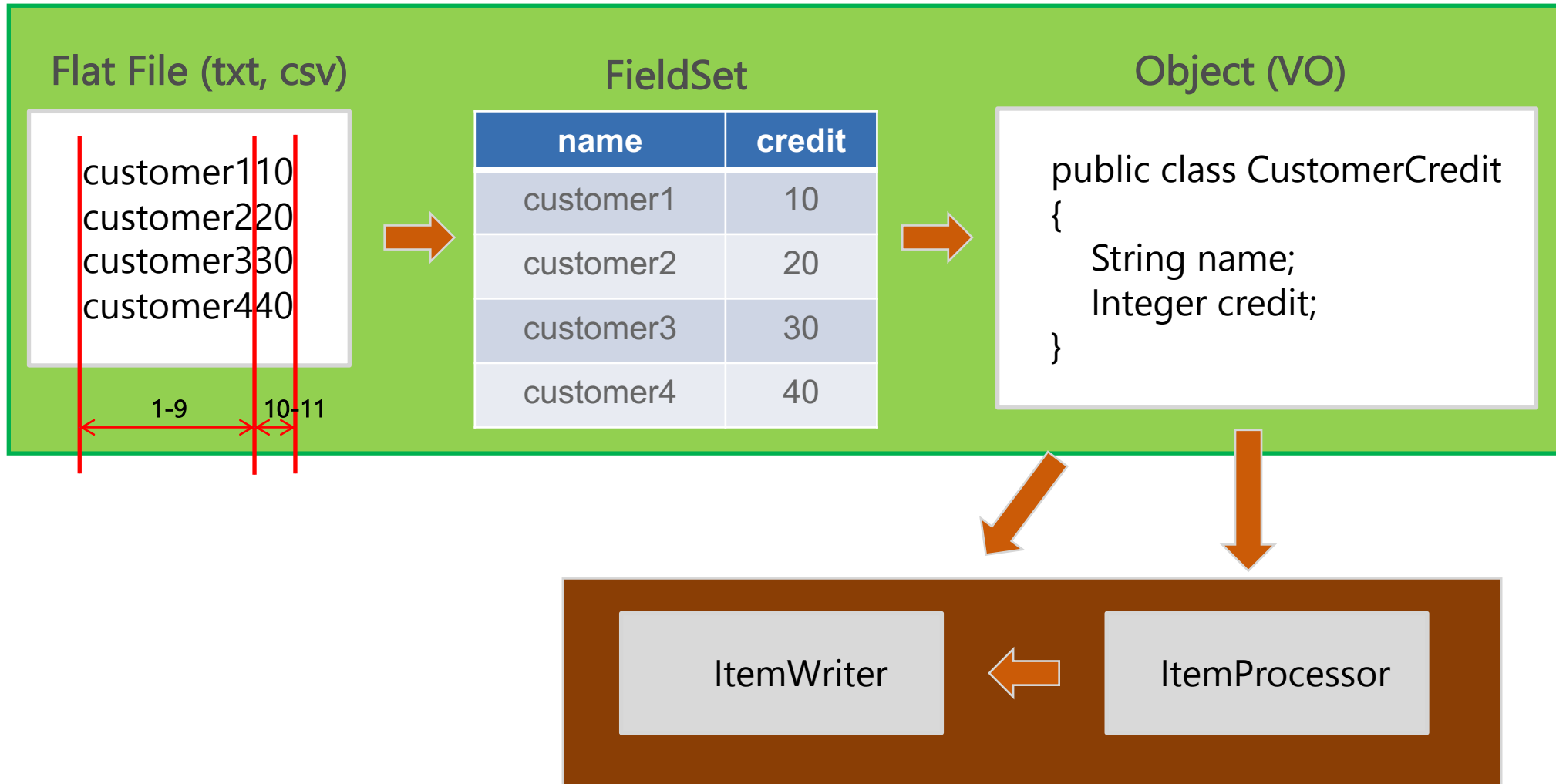


### ❑ Flat File 읽기 설정(Fixed Length 방식)

- FlatFileReader를 통해 txt파일을 한 라인씩 읽은 후에 VO의 field값에 Mapping할 길이를 지정하여 Tokenizing 하고 Tokenizing된 결과를 VO에 Mapping하는 예제

```
<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader">
  <property name="resource" value="data/iosample/input/fixedLength.txt" />
  <property name="lineMapper">
    <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
      <property name="lineTokenizer">
        <bean class="org.springframework.batch.item.file.transform.FixedLengthTokenizer">
          <property name="names" value="name,credit" />
          <property name="columns" value="1-9,10-11" />
        </bean>
      </property>
      <property name="fieldSetMapper">
        <bean class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper">
          <property
            name="targetType" value="org.springframework.batch.sample.domain.trade.CustomerCredit" />
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

❑ Flat File 읽기 설정(Fixed Length 방식)



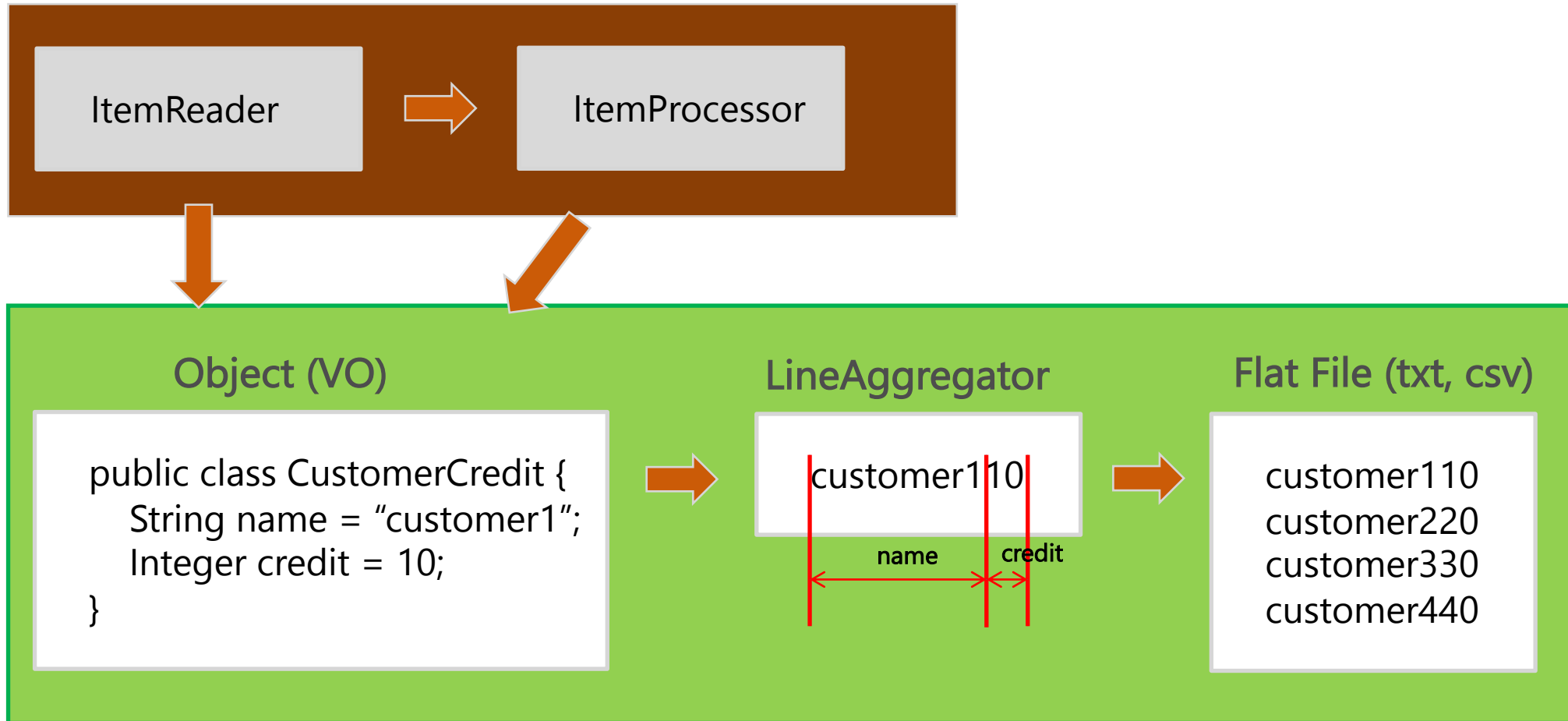
## ❑ Flat File 쓰기 설정(Fixed Length 방식)

- FieldExtractor를 통해 VO의 field값에서 값을 꺼내와서 지정한 Format으로 변환하여 한 라인으로 합친 후에 File에 Write를 수행하는 예제

```
<bean id="itemWriter" class="org.springframework.batch.item.file.FlatFileItemWriter">
  <property name="resource" ref="outputResource" />
  <property name="lineAggregator">
    <bean class="org.springframework.batch.item.file.transform.FormatterLineAggregator">
      <property name="fieldExtractor">
        <bean class="org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor">
          <property name="names" value="name,credit" />
        </bean>
      </property>
      <property name="format" value="%-9s%-2.0f" />
    </bean>
  </property>
</bean>

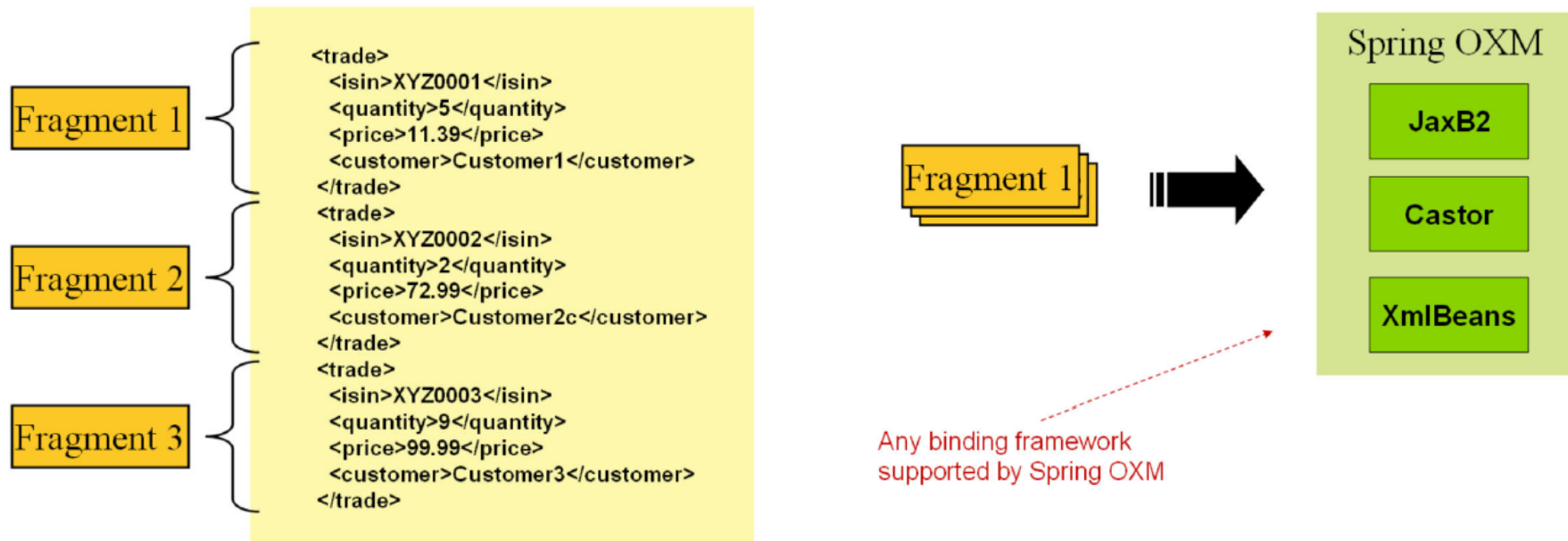
<bean id="outputResource" class="org.springframework.core.io.FileSystemResource">
  <constructor-arg value="target/test-outputs/fixedLengthOutput.txt" />
</bean>
```

❑ Flat File 쓰기 설정(Fixed Length 방식)



## ❑ Xml File 처리 방안

- XML 처리 과정에서 토큰라이징이 필요한 레코드의 행(FieldSet) 대신에 XML 자원을 개별 레코드에 대응되는 'Fragments'의 컬렉션으로 가정.
- Spring Batch는 Fragment를 객체로 바인드 하는데 Object/XML Mapping(OXM)을 사용. 그렇지만 Spring Batch는 특정 XML 바인딩 기술에 종속되지 않음. 대표적인 사용방법은 가장 대중적인 OXM 기술에 대한 일관된 추상화를 제공하는 Spring OXM에 위임.
- Spring OXM에 대한 의존성은 선택이며, 필요 시 Spring Batch에서 특정 인터페이스를 구현하도록 선택 가능.(JDK 6에 기본으로 포함)



## ❑ StaxEventItemReader

- XML레코드 처리 시 설정 항목

설정 항목	설명
fragmentRootElementName	매핑되는 객체를 구성하는 프래그먼트의 루트 엘리먼트 이름.
resource	읽어 들일 데이터의 위치를 지정(파일이나 URL 등)
unmarshaller	XML 프래그먼트를 객체로 매핑하는 Spring OXM에 의해서 제공되는 언마샬링 기능

## ❑ StaxEventItemWriter

- XML레코드 처리 시 설정 항목

설정 항목	설명
resource	작성할 파일의 위치
marshaller	객체를 XML 프래그먼트로 매핑하는 Spring OXM에 의해서 제공되는 마샬링 기능
rootTagName	객체를 XML에 매핑할 때 사용하는 루트 태그 이름.



## □ XML Reader/Writer 설정

```
<bean id="itemReader" class="org.springframework.batch.item.xml.StaxEventItemReader">
  <property name="fragmentRootElementName" value="customer" />
  <property name="resource" value="data/iosample/input/input.xml" />
  <property name="unmarshaller" ref="customerCreditMarshaller" />
</bean>
```

XML파일을 Object로 변환

```
<bean id="itemWriter" class="org.springframework.batch.item.xml.StaxEventItemWriter">
  <property name="resource" ref="outputResource" />
  <property name="marshaller" ref="customerCreditMarshaller" />
  <property name="rootTagName" value="customers" />
  <property name="overwriteOutput" value="true" />
</bean>
```

Object를 XML파일로 변환

```
<bean id="customerCreditMarshaller" class="org.springframework.xml.xstream.XStreamMarshaller">
  <property name="aliases">
    <util:map id="aliases">
      <entry key="customer" value="org.springframework.batch.sample.domain.trade.CustomerCredit" />
      <entry key="price" value="java.math.BigDecimal" />
      <entry key="name" value="java.lang.String" />
    </util:map>
  </property>
</bean>
```

## ❑ Multi File 처리

- N→N: N개의 대상을 읽은 후 읽은 개수 만큼 결과물을 만들어 냄.
- 다수의 파일(**동일 유형의 복수개의 파일**)을 대상으로 **동일 유형의 Batch** 처리 시 사용.
- 하나의 Reader와 Writer를 지정하며 XML과 Flat File 등 다양한 입력 타입을 제공

## ❑ Reader 설정

- MultiResourceItemReader를 통해 1개 이상의 리소스를 읽어온 다음 Reader에게 데이터처리를 위임함.
- input resource경로에 \*를 사용하여 다수의 파일을 처리 가능.

```
<bean id="multiResourceReader" class="org.springframework.batch.item.file.MultiResourceItemReader">  
  <property name="resources" value="classpath:data/multiResourceJob/input/file-*.txt" />  
  <property name="delegate" ref="flatFileItemReader" />  
</bean>
```

## ❑ Writer 설정

- MultiResourceItemWriter를 통해 파일당 출력할 item 개수를 지정한 다음 Writer에게 데이터처리를 위임.

```
<bean id="itemWriter" class="org.springframework.batch.item.file.MultiResourceItemWriter" scope="step">  
  <property name="resource" value="#{jobParameters['output.file.path']}" />  
  <property name="itemCountLimitPerResource" value="6" />  
  <property name="delegate" ref="delegateWriter" />  
</bean>
```

## □ DB 처리

- DB는 Batch 저장 메커니즘의 중심으로 만일 SQL 문이 백만 행을 반환하는 경우 결과 집합은 모든 행을 읽을 때 까지 메모리에 모든 결과를 보유함.
- Spring Batch는 이 문제를 해결하기 위해 Cursor와 Paging기반 Database ItemReader를 제공.

## □ DB처리 방식의 유형

구분	유형	설명
처리건수 축소	Cursor 기반	1라인씩 읽어서 처리
	Paging 기반	Page Size만큼 읽어서 처리
대상의 범위 축소	Driving Query	1단계 : 조회조건을 통해 Primary Key 도출 2단계 : Primary Key를 이용하여 데이터 처리

## □ 처리 가능 DB 유형

구분	설명
JDBC	<ul style="list-style-type: none"> <li>- 가장 기본적인 데이터 접근 지원 프레임워크</li> <li>- DriverManager 클래스, Connection, 다양한 구문 , ResultSet 클래스로 구성</li> <li>- 특징 : 일괄처리 능력</li> </ul>
iBatis	<ul style="list-style-type: none"> <li>- ORM도구로 자바 빈 프로퍼티 값을 생성하는 데 필요한 SQL코드 직접 작성.</li> <li>- 각 데이터베이스에서 표준 SQL을 확장한 기능을 완벽히 사용 가능</li> </ul>
Hibernate	<ul style="list-style-type: none"> <li>- 객체-관계 매핑 도구로 자바 코드에 최소한의 영향을 가하여 DB에 있는 일반 자바 객체를 찾고 저장하고 삭제가 가능</li> </ul>

## □ 배치 업무의 유형

구분	업무 유형	배치 구현 방안
배치전용 프로그램	설계 초기부터 대량(Bulk) 데이터 처리 고려해서 구현	ItemReader ItemWriter
온라인시스템, Thin Client Program	기존의 Service, Dao를 배치업무로 사용	ItemReaderAdapter ItemWriterAdapter

## □ 구현 방안

- targetMethod가 read() 메소드의 형태와 동일해야 함.
- ItemReader 인터페이스를 구현하고 있는 클래스가 아니더라도 메소드 호출 시 하나씩 읽어온 Object를 반환하고 더 읽을 것이 없으면 null을 반환하는 ItemReader의 동작방식만 만족한다면 사용 가능.
- ItemReader의 경우 ItemReaderAdapter를 이용해서 targetObject 속성에서 실행할 클래스의 bean선언을 참조하고 targetMethod속성에서 ItemReader.read()와 같은 역할을 하는 메소드를 지정함.

```
<bean id="itemReader" class="org.springframework.batch.item.adapter.ItemReaderAdapter">
  <property name="targetObject" ref="fooService" />
  <property name="targetMethod" value="generateFoo" />
</bean>
<bean id="fooService" class="org.springframework.batch.item.sample.FooService" />
```

```
<bean id="itemWriter" class="org.springframework.batch.item.adapter.ItemWriterAdapter">
  <property name="targetObject" ref="fooService" />
  <property name="targetMethod" value="processFoo" />
</bean>
<bean id="fooService" class="org.springframework.batch.item.sample.FooService" />
```

1. 개요
2. Batch 구성요소
3. Spring Batch 2.x
4. Spring Batch 3.x
5. Batch Processing
6. Batch Support
  - Scaling and Parallel Processing
  - Listener
7. eGovFrame Batch 제공기능
8. 참고자료

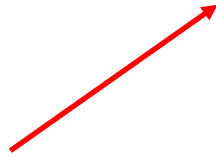
## ❑ Multi-threaded Step

- 하나의 Step을 여러 개의 Thread로 처리하는 방식

**장점** : 동일한 Step을 병렬로 빠르게 실행함

**단점** : 실패시 실패지점부터 이어서 실행하기가 불가능

(참고사항) **thread-safe** 를 지원하는 reader와 writer만 사용 가능



```

*
* <p>
* The implementation is thread-safe in between calls to
* {@link #open(ExecutionContext)}, but remember to use
* <code>saveState=false</code> if used in a multi-threaded client (no restart
* available).
* </p>
*
* @author Thomas Risberg
* @author Dave Syer
* @author Michael Minella
* @since 2.0
*/
public class JdbcPagingItemReader<T> extends AbstractPagingItemReader<T> implements
    private static final String START_AFTER_VALUE = "start.after";

    public static final int VALUE_NOT_SET = -1;

    private DataSource dataSource;
  
```

- Step 구성 요소 중 <tasklet> 속성에 TaskExecutor를 추가하여 구현
- TaskExecutor는 SimpleAsyncTaskExecutor 와 ThreadPoolTaskExecutor 가 있음
  1. SimpleAsyncTaskExecutor : 간단한 테스트용으로 사용 추천 (요청시마다 스레드 생성)
  2. ThreadPoolTaskExecutor : 운영환경에서 사용 추천 (스레드풀내의 자원 활용)

```

<step id="step1">
  <tasklet task-executor="taskExecutor">...</tasklet>
</step>
<bean id="taskExecutor" class="org.springframework.core.task.SimpleAsyncTaskExecutor" />
  
```

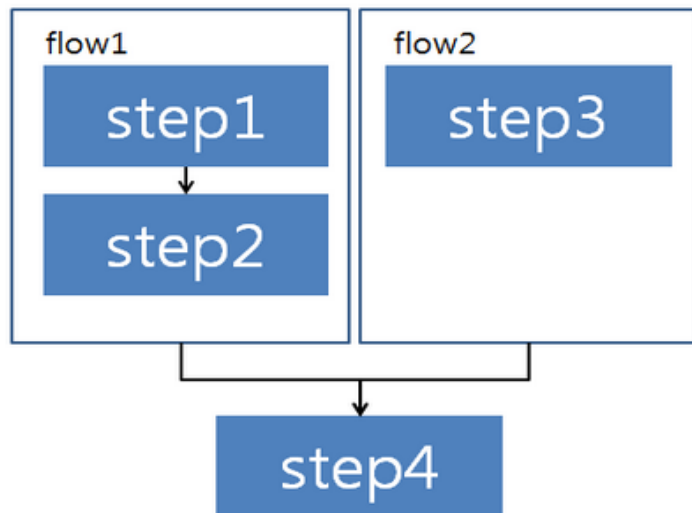
- Thread의 수는 기본값으로 4가 설정되어 있으나 필요하다면 증가시켜 사용 가능

```

<step id="loading">
  <tasklet task-executor="taskExecutor" throttle-limit="10">
...
  </tasklet>
</step>
  
```

## ❑ Parallel Step

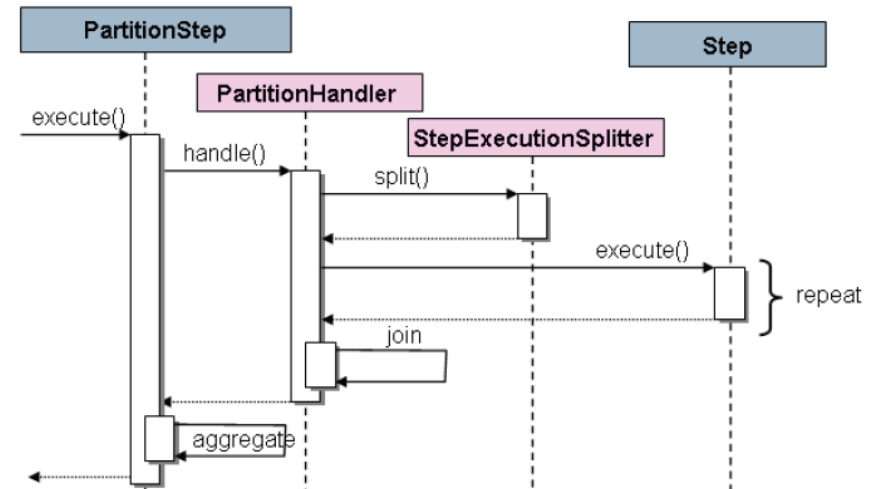
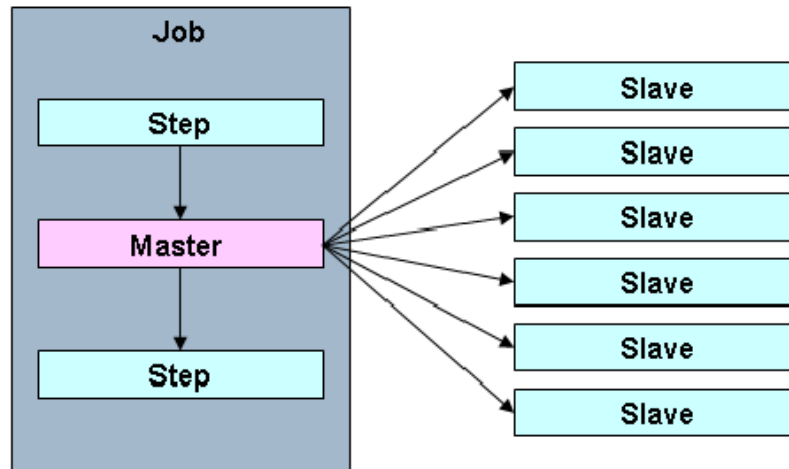
- 병렬화가 필요한 부분에 따라 영역을 나눈 후 각 단계별로 할당하여 하나의 프로세스에서 병렬처리 가능
- 각 작업 분할은 최종 종료 상태로 통합되기 전에 모두 완료하도록 구성해야 함
- 분리된 flow들이 모두 완료되어야만 다음 step으로 진행 가능



```
<job id="job1">
  <split id="split1" task-executor="taskExecutor" next="step4">
    <flow>
      <step id="step1" parent="s1" next="step2" />
      <step id="step2" parent="s2" />
    </flow>
    <flow>
      <step id="step3" parent="s3" />
    </flow>
  </split>
  <step id="step4" parent="s4" />
</job>
<beans:bean id="taskExecutor" class="org.spr...SimpleAsyncTaskExecutor" />
```

## ❑ Partitioning

- 하나의 스텝(Master)을 파티셔너를 이용하여 각각의 Slave로 나누어 동시에 병렬로 실행하는 처리 방법
- Partitioner 인터페이스를 구현해야함. `partition(int gridSize);`
- 100개의 데이터가 있고 `gridSize="2"` 설정할 경우, 2개의 slave로 나누어져 실행됨  
( partition0 : 1~50, partition1 : 51~100 )



설정 항목	설명
Grid-Size	<ul style="list-style-type: none"> <li>- 실행할 Context영역의 개수를 지정하는 단위 (Thread Pool 개수 지정)</li> <li>- Grid-Size만큼의 Context가 생성되며, 각 Context마다 데이터가 나뉘어져 들어감</li> </ul>



## ❑ Listener

- Batch 처리 중 이벤트 발생 시 해당 이벤트를 Intercepting하여 필요한 Logic 수행

## ❑ 사용 예

- Job설정파일에서 <listener> 이용해 등록

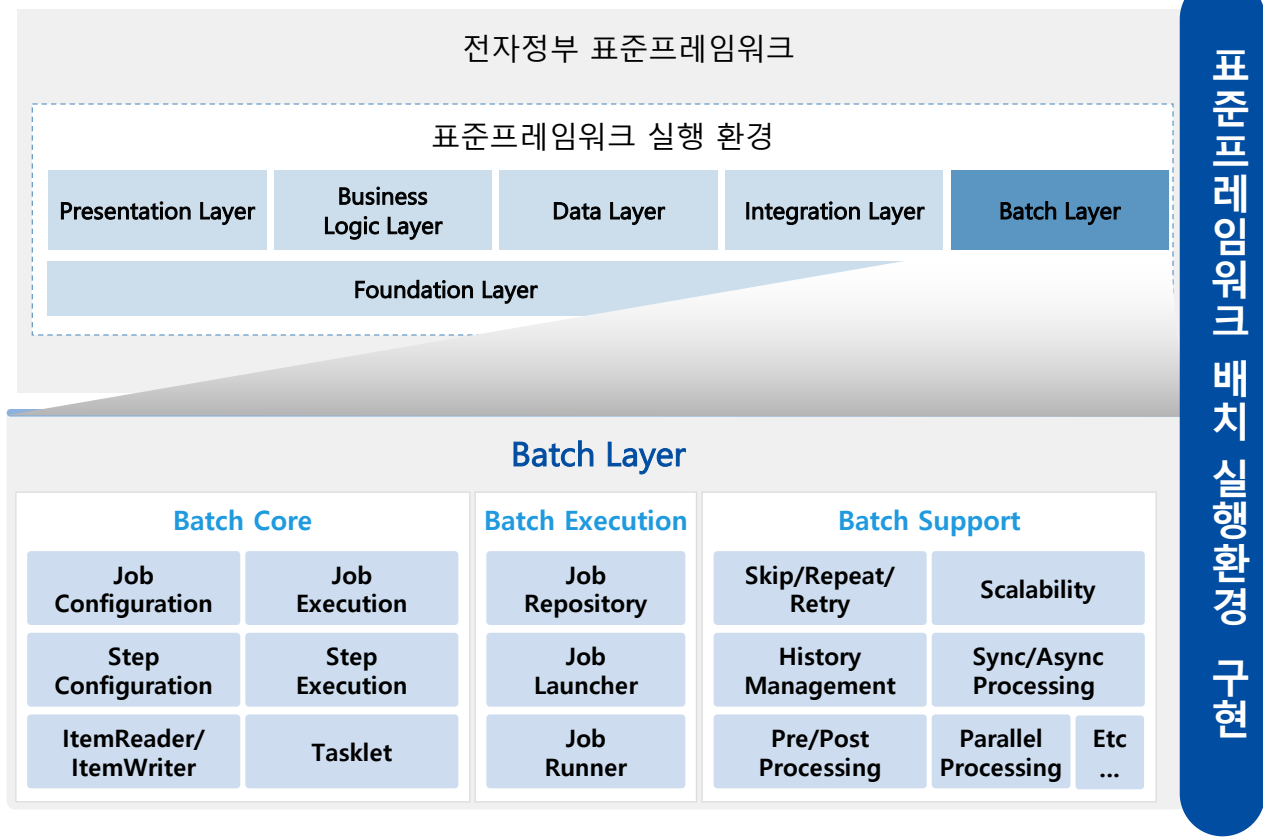
```
<job id="compositeItemWriterJob" xmlns="http://www.springframework.org/schema/batch" >
  <step id="compositeItemWriterStep1">
    <tasklet>
      <chunk reader="fileItemReader" processor="processor" writer="compositeWriter" commit-interval="1">
        <streams>
          <stream ref="fileItemReader"/>
          <stream ref="fileItemWriter1"/>
          <stream ref="fileItemWriter2"/>
        </streams>
      </chunk>
    </tasklet>
    <listeners>
      <listener ref="forStepTestListener" />
    </listeners>
  </step>
  <listeners>
    <listener ref="forJobTestListener" />
  </listeners>
</job>

<bean id="forStepTestListener" class="org.springframework.batch.sample.ForStepTestListener" />
<bean id="forJobTestListener" class="org.springframework.batch.sample.ForJobTestListener" />
```

1. 개요
2. Batch 구성요소
3. Spring Batch 2.x
4. Spring Batch 3.x
5. Batch Processing
6. Batch Support
7. eGovFrame Batch 제공기능
  - 실행환경 배치처리레이어
  - DB 처리
  - Job Runner
  - Support
8. 참고자료

- 표준프레임워크 배치 실행환경은 3개 레이어(Core, Support, Execution Layer)로 구성되며, 일괄 (배치) 처리를 위한 기반 환경을 제공

### 배치 실행환경 구성



### 주요 내용

#### ▶ 배치실행 관리

다양한 배치 실행 기능 지원

- 시작(Run)
- 재시도(Retry)
- 건너뛰기(Skip)
- 재시작(Restart)

#### ▶ 전처리/후처리 관리

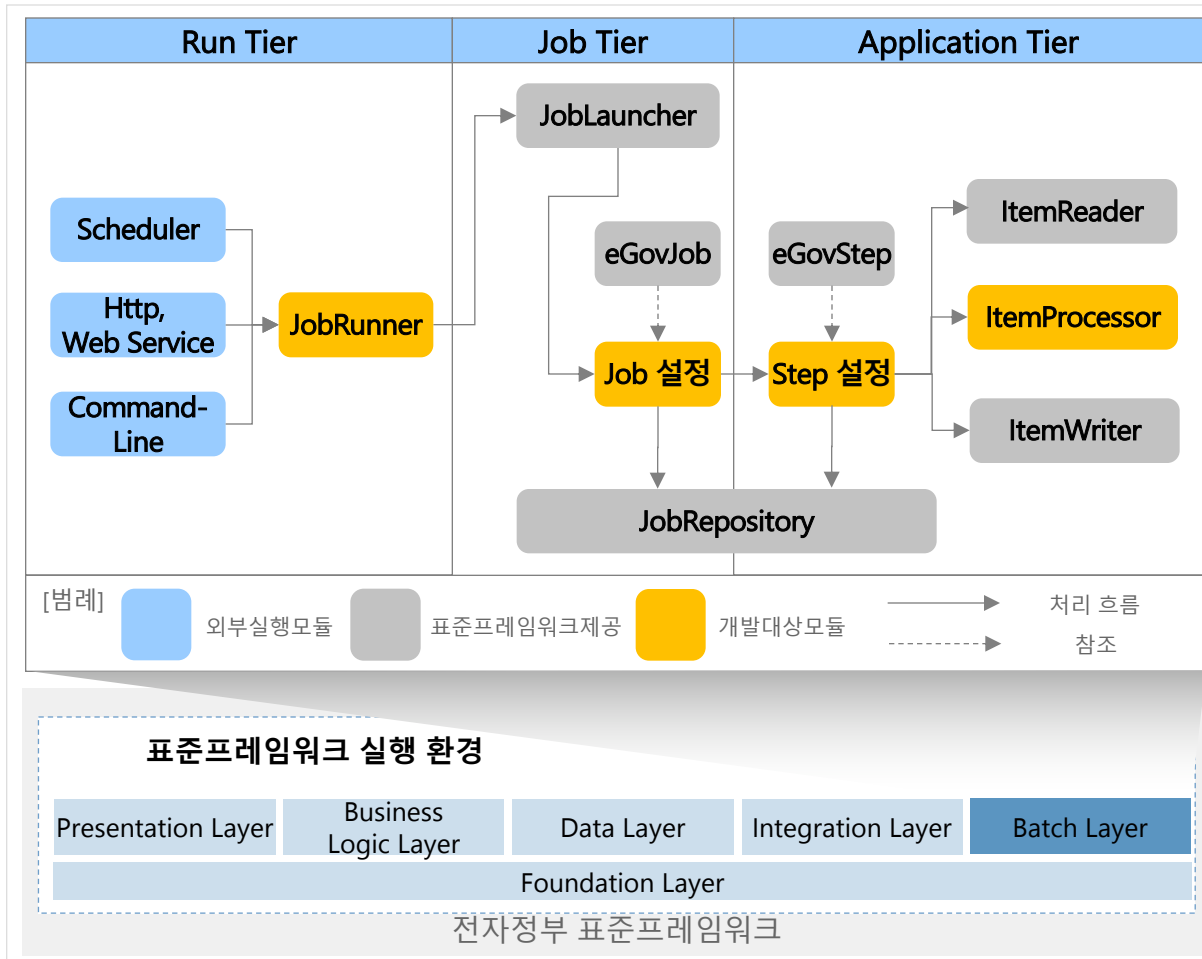
프로세스 전/후 단계에서 이벤트 처리(Event Handling) 기능 제공

#### ▶ 병렬처리

배치 처리 시 멀티스레드 기반의 병렬처리 기능 제공

### ❑ eGovFrame Batch Architecture

- 표준프레임워크 배치 실행환경은 일괄처리 기능 구현에 필요한 사항을 프레임워크 영역에서 제공함으로써 개발자의 변경 부분을 작업 설정 및 비즈니스 구현 등으로 최소화



분류	구성요소	제공 기능
Run Tier	Job Runner	Job 실행 유형 설정
	Job Locator	기 등록된 Job 검색
	Job Configuration	기 등록된 Job 설정
Job Tier	Job Launcher	Job 실행 모듈
	Job	Job 정보 설정
	Job Repository	Job 수행이력 기록
Application Tier	Step	Step 정보 설정
	Item Reader	리소스 형식에 따라 데이터 읽기
	Item Processor	비즈니스 처리
	Item Writer	리소스 형식에 따라 데이터 쓰기

## □ 전자정부 배치 실행환경 구성요소

분류	기술요소	제공 기능
Batch Core	Job Configuration	• Job설정 기능 제공
	Step Configuration	• Step설정 기능 제공
	ItemReader/ItemWriter	• File/DB 처리 기능 제공
	Job Execution	• Job Instance의 수행 기능 제공
	Step Execution	• Step Instance의 수행 기능 제공
	Tasklet	• Job의 실행 중 배치 작업 외 단순처리가 필요한 작업(파일이동 등)의 처리기능을 제공
Batch Execution	Job Repository	• JobExecution, StepExecution 정보를 저장
	Job Runner	• Scheduler, CommandLine (CronTab), Http/Webservice 방식 지원 인터페이스 제공
	Job Launcher	• Job Repository 및 실행 방법 설정 기능 제공
Batch Support	History Management	• JobRepository와 이력관리 기능 제공
	Scalability	• Partitioning 기능 제공
	Sync/Async Processing	• JobLauncher 설정을 통한 동기/비동기 처리 기능 제공
	Parallel Processing	• Job 설정을 사용하여 병렬처리 기능 제공
	Pre/Post Processing	• Listener를 사용한 전처리/후처리 Event Handling 기능 제공
	Skip/Repeat/Retry	• 건너뛰기, 반복, 재시도 기능 제공
	Job·Step·Resource Variable/Code Base Exception/ Shell Processing/File Delete/Index File Reader/Center Cut	• Job·Step·Resource 사용자 정의 변수, 코드 에러 처리, 쉘 실행, 파일삭제, 인덱스 기반 Reader, 센터컷 기능 제공

## ❑ Flat File 처리 - eGovFrame Batch 제공 FlatFileItemReader (1/3)

- 파일 ItemReader의 요소 중 성능 저하 요인인 LineMapper 부분 개선.
- FieldSet을 사용하지 않으므로 Tokens → FieldSet으로 변환하는 과정이 없음 .
- 전자정부 Batch 실행환경에서 제공하는 EgovDefaultLineMapper, EgovLineTokenizer, EgovObjectMapper를 사용하는 경우 Token 상태에서 Object로 직접 맵핑됨 .

개선 항목	설명
EgovDefaultLineMapper	EgovLineTokenizer와 EgovObjectMapper가 변경됨에 따라 LineMapper 총 과정을 제어하는 DefaultLineMapper를 변경하여 EgovDefaultLineMapper 제공
EgovLineTokenizer	전자정부에서는 FieldSet을 사용하지 않기때문에 FieldSet을 반환하는 LineTokenizer 인터페이스를 변경하여 EgovLineTokenizer 제공
EgovAbstractLineTokenizer	LineTokenizer 인터페이스가 EgovLineTokenizer로 변경됨에 따라 토큰나이징만 관여하는 추상 클래스 EgovAbstractLineTokenizer 제공
EgovDelimitedLineTokenizer	Spring에서 제공하는 DelimitedLineTokenizer의 성능을 개선한 EgovDelimitedLineTokenizer 제공
EgovObjectMapper	전자정부에서는 FieldSet을 사용하지 않고 토큰나이징 된 값들을 직접 Object에 맵핑하는 EgovObjectMapper를 제공

## ❑ Flat File 처리 (Delimited) - eGovFrame Batch 제공 FlatFileItemReader (2/4)

- 읽어 들인 문자열에서 구분자를 경계값으로 사용하여 필드를 분리

```
<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step">
  <property name="resource" value="#{jobParameters[inputFile]}" />
  <property name="lineMapper">
    <bean class="egovframework.rte.bat.core.item.file.mapping.EgovDefaultLineMapper">
      <property name="lineTokenizer">
        <bean class="egovframework.rte.bat.core.item.file.transform.EgovDelimitedLineTokenizer">
          <property name="delimiter" value="," />
        </bean>
      </property>
      <property name="objectMapper">
        <bean class="egovframework.rte.bat.core.item.file.mapping.EgovObjectMapper">
          <property name="type" value="egovframework.brte.sample.domain.trade.CustomerCredit" />
          <property name="names" value="name,credit" />
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

설정항목	내용	예시
delimiter	필드의 경계를 구별해주는 문자	, (кома)
type	VO 클래스	org.springframework.batch.CustomerCredit
names	VO 클래스의 필드	name,credit

## ❑ Flat File 처리 (FixedLength) - eGovFrame Batch 제공 FlatFileItemReader (3/4)

읽어 들인 문자열에서 필드의 경계를 파일 내의 문자열 길이로 판단하여 필드를 분리

```
<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step">
  <property name="resource" value="#{jobParameters[inputFile]}" />
  <property name="lineMapper">
    <bean class="egovframework.rte.bat.core.item.file.mapping.EgovDefaultLineMapper">
      <property name="lineTokenizer">
        <bean class="egovframework.rte.bat.core.item.file.transform.EgovFixedLengthTokenizer">
          <property name="column" value="1-9,10-11" />
        </bean>
      </property>
      <property name="objectMapper">
        <bean class="egovframework.rte.bat.core.item.file.mapping.EgovObjectMapper">
          <property name="type" value="egovframework.brte.sample.domain.trade.CustomerCredit" />
          <property name="names" value="name,credit" />
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

설정항목	내용	예시
column	필드 경계의 범위	1-9,10-11
type	VO 클래스	org.springframework.batch.CustomerCredit
names	VO 클래스의 필드	name,credit



## ❑ Flat File 처리 (ByteLength) - eGovFrame Batch 제공 FlatFileItemReader (4/4)

- EgovFixedByteLengthTokenizer는 FixedLengthTokenizer와 유사하나, **byte 문자열을 기준으로** 필드의 경계값을 구해 필드를 분리

```
<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step">
  <property name="resource" value="#{jobParameters[inputFile]}" />
  <property name="lineMapper">
    <bean class="egovframework.brte.core.item.file.mapping.EgovDefaultLineMapper">
      <property name="lineTokenizer">
        <bean class="egovframework.brte.core.item.file.transform.EgovFixedByteLengthTokenizer">
          <property name="encoding" value="utf-8"/>
          <property name="columns" value="1-9,10-11" />
        </bean>
      </property>
      <property name="objectMapper">
        <bean class="egovframework.brte.core.item.file.file.mapping.EgovObjectMapper">
          <property name="type" value="egovframework.brte.sample.domain.trade.CustomerCredit" />
          <property name="names" value="name,credit" />
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

설정항목	내용	예시
column	필드 경계의 길이	1-9,10-11
encoding	byte 문자열 인코딩 타입	utf-8
type	VO 클래스	org.springframework.batch.CustomerCredit
names	VO 클래스의 필드	name,credit

❑ Flat File 처리- eGovFrame Batch 제공 FlatFileItemWriter (1/3)

- Spring Batch의 FileItemWriter 개선사항

개선험목	설명
BeanWrapperFieldExtractor 성능 개선	item에서 field 값을 추출하는 과정의 성능을 개선한 EgovFieldExtractor 제공
FormatterLineAggregator 경량화	FormatterLineAggregator 를 경량화하여 고정길이 방식의 EgovFixedLineAggregator 제공

❑ Flat File 처리 ( FixedLength ) - eGovFrame Batch 제공 FlatFileItemWriter (1/3)

```
<bean id="itemWriter" class="org.springframework.batch.item.file.FlatFileItemWriter" scope="step">
  <property name="resource" value="#{jobParameters[outputFile]}" />
  <property name="lineAggregator">
    <bean class="egovframework.brte.core.item.file.transform.EgovFixedLengthLineAggregator">
      <property name="fieldExtractor">
        <bean class="egovframework.brte.core.item.file.transform.EgovFieldExtractor">
          <property name="names" value="name,credit" />
        </bean>
      </property>
      <property name="fieldRanges" value="9,2" />
    </bean>
  </property>
</bean>
```

EgovFlatFileItemWriter 설정항목 (FixedLength)	설명
fieldRanges	Item의 필드 값들을 1 Line의 String으로 만들 때 필드값의 범위(고정길이) 지정
names	VO 클래스의 필드를 나타낸다.

## ❑ Flat File 처리 (Delimited) - eGovFrame Batch 제공 FlatFileItemWriter (1/3)

```
<bean id="itemWriter" class="org.springframework.batch.item.file.FlatFileItemWriter" scope="step">
  <property name="resource" value="#{jobParameters[outputFile]}" />
  <property name="lineAggregator">
    <bean class="org.springframework.batch.item.file.transform.DelimitedLineAggregator">
      <property name="fieldExtractor">
        <bean class="egovframework.brte.core.item.file.transform.EgovFieldExtractor">
          <property name="names" value="name,credit" />
        </bean>
      </property>
      <property name="delimiter" value="," />
    </bean>
  </property>
</bean>
```

EgovFlatFileItemWriter 설정항목 (Delimited )	설명
delimiter	Item의 필드 값들을 1 Line의 String으로 만들 때 경계가 되는 구분자 지정
names	VO 클래스의 필드를 나타낸다.

## □ DB 처리 - eGovFrame Batch 제공 DB ItemWriter

- Spring에서 제공하는 JdbcBatchItemWriter는 상용 Batch F/W와 비교하여 대용량 데이터 처리 속도 차이 발생
- JdbcBatchItemWriter에서는 사용자가 PreparedStatement를 setter하기 위한 클래스를 직접 작성하지 않으며 XML 설정시 쿼리의 파라미터 값을 지정하여 자동으로 PreparedStatement를 setter해주는 기능 제공
- Spring Batch 설정 : sql작성 시 필드명에 ":"를 붙여서 작성

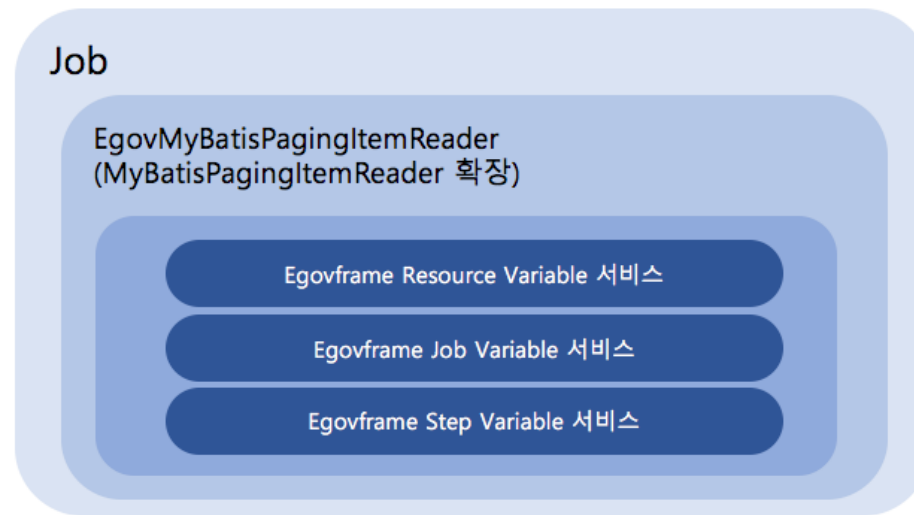
```
<bean id="empDBBatchWriter" class="org.springframework.batch.item.database.JdbcBatchItemWriter">
    <property name="assertUpdates" value="true" />
    <property name="itemSqlParameterSourceProvider">
        <bean class=".....BeanPropertySqlParameterSourceProvider" />
    </property>
    <property name="sql"
        value="insert into UIP_EMPLOYEE (num, name, sex) values (:num, :name, :sex)" />
    <property name="dataSource" ref="dataSource" />
</bean>
```

- 전자정부 배치 설정 : sql작성 시 ?로 설정하고 필드명을 params에 작성

```
<bean id="egovJdbcBatchItemWriter3" class="org.springframework.batch.item.database.EgovJdbcBatchItemWriter">
    <property name="assertUpdates" value="true" />
    <property name="itemPreparedStatementSetter">
        <bean class=".....EgovMethodMapItemPreparedStatementSetter" />
    </property>
    <property name="sql" value="insert into UIP_EMPLOYEE (num, name, sex) values (?, ?, ?)" />
    <property name="params" value="num, name, sex"/>
    <property name="dataSource" ref="dataSource" />
</bean>
```

## □ DB 처리 - eGovFrame Batch 제공 Mybatis Paging Item Reader

- Mybatis를 Pageing 기반으로 처리 하는 Reader 기능
- 배치 처리시 Mybatis 처리 과정 중 Job, Step, Resouce Variable 사용자 정의 변수 사용 가능
- MybatisPagingItemReader를 확장한 EgovMybatisPagingItemReader 서비스



### - EgovMybatisPagingItemReader 설정

```
<bean id="mybatisJobStep.mybatisItemReader" class="egovframework.rte.bat.item.database.EgovMybatisPagingItemReader" scope="step">
    <property name="sqlSessionFactory" ref="sqlSession" />
    <property name="resourceVariable" ref="resourceVariable" />
    <property name="jobVariable" ref="jobVariable" />
    <property name="stepVariable" ref="stepVariable" />
    <property name="queryId" value="EmpMapper.selectEmpList" />
    <property name="pageSize" value="#{100}" />
</bean>
```

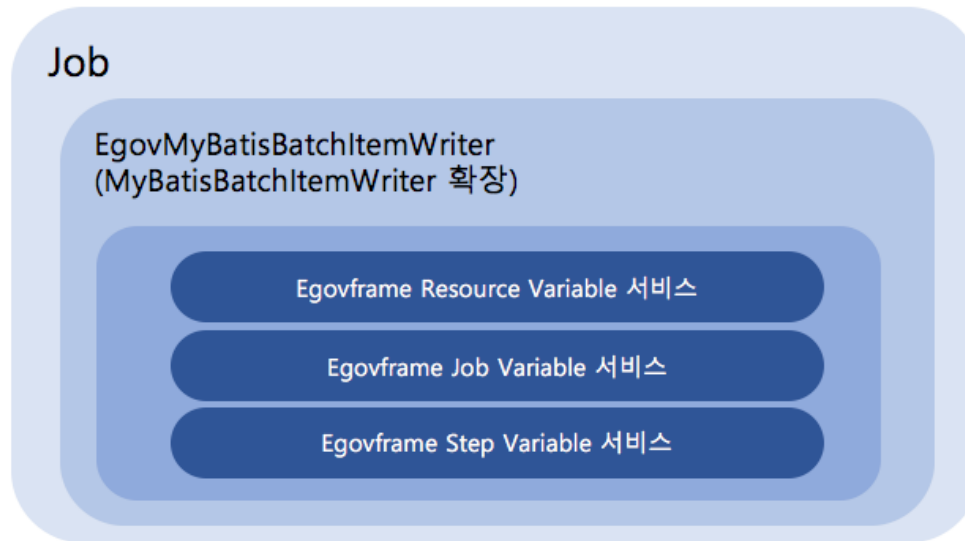
## □ DB 처리 - eGovFrame Batch 제공 Mybatis Paging Item Reader

### – EgovMyBatisPagingItemReader 설정항목

설정항목	내용	비고
sqlSessionFactory	reader에 별도로 구현한 sessionFactory	sqlSession
parameterValues	파라미터 전달을 위한 설정	parameterValues
queryId	네임스페이스를 가진 매퍼 파일을 Query Id.	EmpMapper.selectEmpList
scope	해당 Reader가 적용될 Bean Scope	step, job
pageSize	배치가 처리할 페이지 사이즈 크기	{100}
resourceVariable	표준프레임워크 실행환경 Resource Variable 서비스를 사용하기 위한 설정	resourceVariable
jobVariable	표준프레임워크 실행환경 Step Variable 서비스를 사용하기 위한 설정	jobVariable
stepVariable	표준프레임워크 실행환경 Job Variable 서비스를 사용하기 위한 설정	stepVariable

### □ DB 처리 - eGovFrame Batch 제공 Mybatis Paging Item Writer

- Mybatis를 Paging 기반으로 처리 하는 Writer 기능
- 배치 처리시 Mybatis 처리 과정 중 Job, Step, Resouce Variable 사용자 정의 변수 사용 가능
- MybatisPagingItemWriter를 확장한 EgovMybatisPagingItemWriter 서비스



#### - EgovMybatisPagingItemWriter 설정

```
<bean id="mybatisJobStep.mybatisItemWriter" class="egovframework.rte.bat.item.database.EgovMybatisBatchItemWriter">
    <property name="resourceVariable" ref="resourceVariable" />
    <property name="jobVariable" ref="jobVariable" />
    <property name="stepVariable" ref="stepVariable" />
    <property name="sqlSessionFactory" ref="sqlSession" />
    <property name="statementId" value="EmpMapper.updateEmp" />
</bean>
```



## ❑ DB 처리 - eGovFrame Batch 제공 MyBatis Paging Item Writer

– EgovMyBatisPagingItemWriter 설정항목

설정항목	내용	비고
sqlSessionFactory	reader에 별도로 구현한 sessionFactory	sqlSession
statementId	네임스페이스를 가진 매퍼 파일을 Query Id	EmpMapper.selectEmpList
resourceVariable	표준프레임워크 실행환경 Resource Variable 서비스를 사용하기 위한 설정	resourceVariable
jobVariable	표준프레임워크 실행환경 Step Variable 서비스를 사용하기 위한 설정	jobVariable
stepVariable	표준프레임워크 실행환경 Job Variable 서비스를 사용하기 위한 설정	stepVariable

## □ Batch Runner 유형

- 전자정부 배치 실행환경에서는 작업실행 유형에 따라 미리 JobRunner를 구현한 표준 Batch Runner를 제공

유형	설명
EgovBatchJobRunner	Web, Java Application 등을 이용하여 범용적으로 실행되는 일괄처리 작업에 사용.
EgovCommandLineRunner	외부 프로그램(Windows: / Unix/Linux: crontab 등)이나 명령 프롬프트(Windows: bat / Unix/Linux: Shell)에서 독립적으로 실행되는 Batch작업에 사용.
EgovSchedulerRunner	주기적으로 실행되어야 하는 일괄처리 작업에 사용.

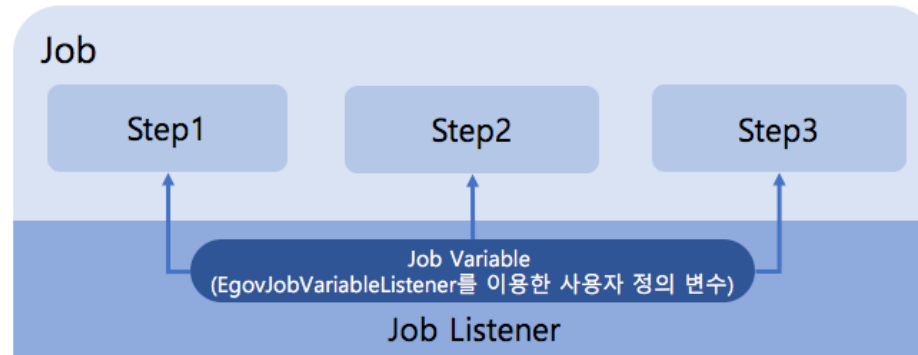
## □ 유형별 Batch Runner 제공 기능

유형	Java Application 실행	Web 실행	Job 상태 모니터링	Scheduling 기능	명령 프롬프트 연동 지원
EgovBatchJobRunner	○	○	○	X	△
EgovCommandLineRunner	○	X	○	X	○
EgovSchedulerRunner	○	○	X	○	△

- EgovBatchRunner, EgovSchedulerRunner에서 명령 프롬프트 연동을 위해서는 추가적인 구현이 필요

## ❑ Support - eGovFrame Batch 제공 Job Variable

- 변수 선언 후 Job Listeners를 통해서 Job에서 사용자 정의 변수를 사용 할 수 있는 기능
- Job, Step, Tasklet, ItemReader, ItemWriter, ItemProcess 서비스에서 사용가능



### - Job Variable 설정

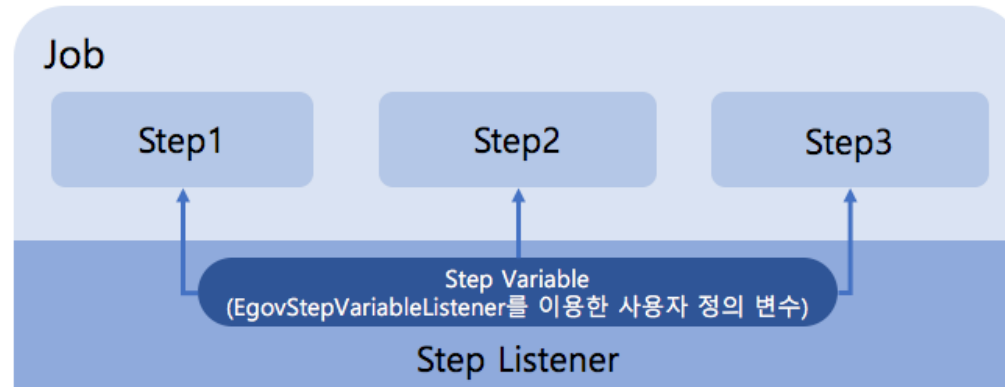
```
<bean id="egovJobVariableListener" class="egovframework.rte.bat.support.EgovJobVariableListener">
  <property name="pros"><props>
    <prop key="JobVariableKey1">JobVariableValue1 </prop>
    <prop key="JobVariableKey2">JobVariableValue2 </prop>
    <prop key="JobVariableKey3">JobVariableValue3 </prop>
  </props> </property>
</bean>
```

### - Job Listener 설정

```
<job id="delimitedToDelimitedJob-JobVariable" parent="eGovBaseJob" xmlns="http://www.springframework.org/schema/batch"> <listeners>
  <listener ref="egovJobVariableListener" />
</listeners>
<step id="step1">
  <tasklet ref="taskletJob" />
</step>
</job>
```

## ❑ Support - eGovFrame Batch 제공 Step Variable

- 변수 선언 후 Step Listeners를 통해서 Step에서 사용자 정의 변수를 사용 할 수 있는 기능
- Step, Tasklet, ItemReader, ItemWriter, ItemProcess 서비스에서 사용가능



### - Step Variable 설정

```
<bean id="egovStepVariableListener" class="egovframework.rte.bat.support.EgovStepVariableListener">
  <property name="pros"><props>
    <prop key="StepVariableKey1">StepVariableValue1</prop>
    <prop key="StepVariableKey2">StepVariableValue2</prop>
    <prop key="StepVariableKey3">StepVariableValue3</prop>
  </props></property>
</bean>
```

### - Step Listener 설정

```
<job id="delimitedToDelimitedJob-StepVariable" parent="eGovBaseJob" xmlns="http://www.springframework.org/schema/batch">
  <step id="step1">
    <tasklet ref="taskletStep" />
    <listeners><listener ref="egovStepVariableListener" /></listeners>
  </step>
</job>
```

### ❑ Support - eGovFrame Batch 제공 ResourceVariable

- Step, Job에서 사용자 정의 Resource Variable 변수를 사용 할 수 있는 기능
- Job, Step, Tasklet, ItemReader, ItemWriter, ItemProcess 서비스에서 사용가능



#### - Resource Variable 설정

```
<bean id="egovResourceVariable" class="egovframework.rte.bat.support.EgovResourceVariable">
  <property name="pros">
    <props>
      <prop key="input.resource">file:./egovframework/batch/data/inputs/csvData.csv</prop>
      <prop key="writer.resource">file:./target/test-outputs/csvOutput.csv</prop>
    </props>
  </property>
</bean>
```

## ❑ Support - eGovFrame Batch 제공 Code Base Exception

- 기존 Message 기반 에러처리는 체계적 메시지 관리, 실시간 변경, 접근성 단점 발생
- 에러처리시 Code Base Exception을 도입하여 에러메세지키를 통하여 에러를 처리가 가능하기 때문에 에러 효율성 향상
- Code Base Exception 데이터 베이스 설정

```
CREATE TABLE BATCH_EXCEPTION_MESSAGE (
    EX_ID BIGINT NOT NULL PRIMARY KEY,
    EX_KEY VARCHAR(255) NOT NULL,
    EX_MESSAGE VARCHAR(2500) NOT NULL
);
INSERT INTO BATCH_EXCEPTION_MESSAGE VALUES(1,'EGOVBATCH000001','배치실행 중 업무 관련 에러가 발생 하였습니다.');
```

```
INSERT INTO BATCH_EXCEPTION_MESSAGE VALUES(2,'EGOVBATCH000002','배치실행 중 알수 없는 오류가 발생 하였습니다.');
```

- Code Base Exception 사용

```
try{
    ...
}catch(Exception e){
    throw new EgovBatchException(dataSource,"EGOVBATCH000001");
    //Sql 설정시 EgovBatchException 생성자 파라미터 추가
    //throw new EgovBatchException(dataSource,"EGOVBATCH000001","SELECT EX_MESSAGE FROM
    BATCH_EXCEPTION_MESSAGE WHERE EX_KEY = ?"); }
```

## □ Support - eGovFrame Batch 제공 Shell Processing

- 배치 실행 중 요구되는 외부 실행(MacOs/Unix/Linux Shell, Windows Command)에 대한 문제점 발생
- 실시간 외부 실행으로 시스템 작업 및 다른 배치와 호환성 증대
- Shell Processing 설정

```
<bean id="egovJobVariableListener" class="egovframework.rte.bat.support.EgovJobVariableListener">
  <property name="pros">
    <props>
      <!-- # Windows의 경우 사용 -->
      <prop key="encoding">MS949</prop>
      <prop key="shellScript"><![CDATA[
        echo "Shell Script Execution Completed !!!"
        exit 0 ]]></prop>
      <!-- # Unix, Linux, MacOS 등의 경우 사용 -->
      <!-- <prop key="encoding">UTF-8</prop>
      <prop key="shellScript"><![CDATA[
        echo "Shell Script Execution Completed !!!"
        echo exit 0]]> </prop> -->
    </props>
  </property>

  <bean id="taskletShell" class="egovframework.rte.bat.core.step.TaskletShellStep" scope="step">
    <property name="shellScript" value="#{jobExecutionContext[shellScript]}" />
    <property name="encoding" value="#{jobExecutionContext[encoding]}" />
  </bean>
```

설정항목	내용	비고
shellScript	실행 할 스크립트	
encoding	문자 인코딩 방식	UTF-8 또는 MS949

## ❑ Support - eGovFrame Batch 제공 File Delete

- 배치 실행 중 요구되는 특정 파일 또는 모든파일을 삭제 하기 위한 작업 발생
- TaskletDeleteStep 구현체를 통해 단일 및 모든 파일 삭제
- File Delete 설정

```
<bean id="taskletDelete" class="egovframework.rte.bat.core.step.TaskletDeleteStep" scope="step">
    <!-- 특정 디렉토리의 모든 파일 삭제 시 -->
    <property name="directory" value="file:<dictory fullpath>₩" />
    <!-- 특정 단일 파일 삭제 시 -->
    <property name="directory" value="file:<dictory fullpath>₩<file name>.<file extention>" />
</bean>
```

설정항목	구분	설명
directory	디렉토리 설정	예) <property name="directory" value="file:<dictory fullpath>\\" />
	단일 파일 설정	예) <property name="directory" value="file:<dictory fullpath>\<file name>.<file extention>" />

### - File Delete 사용

```
<bean id="egovJobDeleteStepListener" class="egovframework.rte.bat.support.EgovJobVariableListener" />
<job id="delimitedToDelimitedJob-DeleteStep" parent="eGovBaseJob" xmlns="http://www.springframework.org/schema/batch">
    <listeners>
        <listener ref="egovJobDeleteStepListener" />
    </listeners>
    <step id="stepDelete">
        <tasklet ref="taskletDelete" />
    </step>
</job>
```



## ❑ Support - eGovFrame Batch 제공 Index File Reader

- 배치 Job 정의 시 Resource 엘리먼트의 shell step에 shell script에 포함된 파일명에서 일련번호(index)를 사용할 수 있는 Reader 제공
- Index 파일명을 사용하면 파일의 일련번호를 기준으로 동적인 파일명 생성이 가능 제공
- Index(NDX) Reader 방식 설정

```
<bean id="fileIndex-delimitedItemReader" class="egovframework.rte.bat.core.item.file.EgovIndexFileReader">
    <property name="indexResource" value="file:./src/main/resources/egovframework/batch/data/inputs/csvData_NDX(0)" />
    <property name="lineMapper">
        <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
            <property name="lineTokenizer">
                <bean
class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer">
                    <property name="delimiter" value="," />
                    <property name="names" value="name,credit" />
                </bean>
            </property>
            <property name="fieldSetMapper">
                <bean
class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper">
                    <property name="targetType"
value="egovframework.example.bat.domain.trade.CustomerCredit" />
                </bean>
            </property>
        </bean>
    </property>
</bean>
```

## ❑ Support - eGovFrame Batch 제공 Index File Reader

### – Index(NDX) Reader 방식 치환 로직

NDX File : 파일 이름이 "[이름]\_NDX\_[YYYYMMDDhhmmss]" 형식으로 이루어진 파일  
ex) Sample\_NDX\_20121126151237

NDX 일련번호 : 파일명 끝에 14자리 수의 생성시간(년월일시분초)

NDX 파일명 치환 : "[이름]\_NDX(Index)" 형식의 파일명은 해당 디렉터리의 NDX 파일에 대해 Index에 해당하는 실제 파일명으로 치환됨

(-2) : 일련번호 기준 마지막 파일에서 두 번째 이전 파일명으로 치환됨

(-1) : 일련번호 기준 마지막 파일에서 첫 번째 이전 파일명으로 치환됨

( 0 ) : 일련번호 기준 마지막 파일명으로 치환됨

(+1) : 일련번호 기준 마지막 파일에서 Index를 1 증가시켜 새로운 파일 생성

### – NDX 파일목록 중 잘못된 파일명이 존재할 경우 에러를 발생한다.

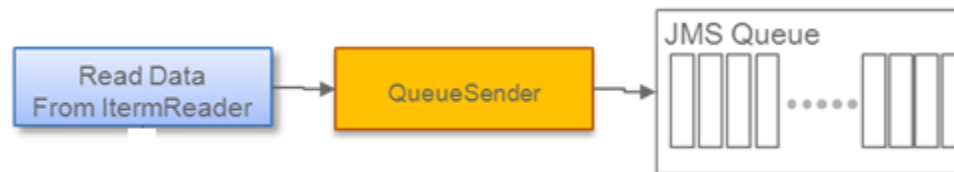
구분	예시	비고
에러	Sample_NDX_20180104	index 자릿수가 10자리
에러	Sample_NDX_000A	index는 숫자만 허용함
무시	Sample_20180104123456	NDX 파일이 아닌 일반 파일로 인식

### ❑ Support - eGovFrame Batch 제공 Center Cut

- 배치 실행 중 요구되는 대용량 데이터 처리를 위해 센터 컷 방식의 작업 발생
- Center Cut은 큐(Queue)를 사용하여 대용량 데이터 처리를 위해 센터 컷 방식의 배치 작업수행을 위한 기능
- Unordered List Item기본적으로 센터컷의 구조는 큐(Queue)를 이용하는 부분을 제외하고는 배치 프로그램과 유사
- Unordered List Item처음 ItemReader를 사용하여 데이터를 읽고 큐에 넣은 Center-Cut Reading Step과, 읽어온 데이터를 가공 후 QueueSender를 통해 Queue에 넣는 구조



- Center-Cut Process Step은 큐에서 들어온 데이터를 읽고 처리 모듈(Business Proc)를 활용하여 데이터를 처리하는 구조



- Center Cut 처리시 큐전송(QueueSender) -> 큐서버(ActiveMQ) 적재 -> 큐수신(QueueReceiver)을 통해 처리되고 센터컷을 단말에서 최종 처리

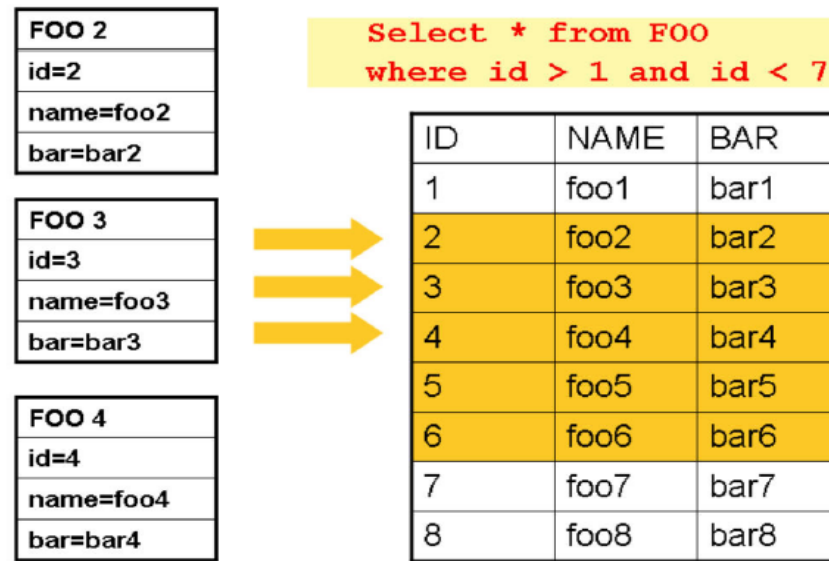
1. 개요
2. Batch 구성요소
3. Spring Batch 2.x
4. Spring Batch 3.x
5. Batch Processing
6. Batch Support
7. eGovFrame Batch 제공기능

### 8. 참고자료

- DB 처리 – Cursor 기반
- DB 처리 – Paging 기반
- DB 처리 – Driving Query 기반

## ❑ Cursor 기반

- Spring Batch에서 Cursor는 Cursor를 초기화해서 열어주는 ItemReader에 기반하며 read가 호출될 때마다 Cursor를 다음 행으로 이동시키고 처리 과정 중에 사용되는 맵핑된 객체를 반환.
- 주의점 : ResultSet.HOLD\_CURSORS\_OVER\_COMMIT 을 지원하기 위해서는 JDBC3.0 드라이버 필요.



- 'FOO' 테이블은 ID, NAME, BAR 3개의 컬럼으로 구성되어 있으며 SQL문을 통해 ID가 1보다 크고 7보다 작은 행의 결과를 조회
- Cursor는 ID 2에서 시작하며 read()가 호출될 때마다 FOO 객체로 매핑되고 Cursor는 다음 행으로 이동

## ❑ Cursor 기반 ItemReader 종류

- JdbcCursorItemReader : Cursor 기반 기술의 JDBC를 구현한 ItemReader
- HibernateCursorItemReader : Hibernate 사용 이슈 원인인 cache나 dirty checking을 제거한 ItemReader로 HQL문을 선언하고 SessionFactory를 전달하는 방식으로 동작함

## ❑ Cursor 기반 ItemReader 설정 : JdbcCursorItemReader

```
<bean id="itemReader" class="org.springframework.batch.item.database.JdbcCursorItemReader">
  <property name="dataSource" ref="dataSource"/>
  <property name="sql" value="select ID, NAME, CREDIT from CUSTOMER"/>
  <property name="rowMapper">
    <bean class="org.springframework.batch.sample.domain.trade.internal.CustomerCreditRowMapper"/>
  </property>
</bean>
```

설정 항목	설명
dataSource	DB connection을 넣어올 수 있는 dataSource를 지정
sql	실행할 쿼리
rowMapper	ResultSet에서 객체를 매핑하는 클래스로 RowMapper 인터페이스를 구현한 클래스 정의

## ❑ Paging 기반 처리

- 데이터베이스 커서를 사용하는 대신 여러번 쿼리를 실행할 수 있는데 실행되는 각 쿼리는 정해진 크기인 페이지만큼의 결과를 가져올 수 있음.
- 실행되는 각 쿼리는 시작 행 번호를 지정하고 페이지에 반환시키고자 하는 행의 수를 지정한 후 사용.

## ❑ Paging 기반 ItemReader① - JdbcPagingItemReader

- 페이지를 형성하는 행을 반환하는데 사용하는 SQL 쿼리를 제공할 책임을 지고 있는 PagingQueryProvider 인터페이스 필요
- 데이터 베이스 유형별로 지원하는 서로 다른 PagingQueryProvider를 사용

```
<bean id="itemReader" class="org.springframework.batch.item.database.JdbcPagingItemReader">
  <property name="dataSource" ref="dataSource"/>
  <property name="queryProvider">
    <bean class="org.springframework.batch.item.database.sql.JdbcPagingQueryProviderFactoryBean">
      <property name="selectClause" value="select id, name, credit"/>
      <property name="fromClause" value="from customer"/>
      <property name="whereClause" value="where status=:status"/>
      <property name="sortKey" value="id"/>
    </bean>
  </property>
  <property name="parameterValues">
    <map><entry key="status" value="NEW"/></map>
  </property>
  <property name="pageSize" value="1000"/>
  <property name="rowMapper" ref="customerMapper"/>
</bean>
```

### ❑ Paging 기반 ItemReader② - JpaPagingItemReader

- JPA는 하이버네이트 StatelessSession과 비슷한 개념이 없으므로 JPA 명세에서 제공하는 다른 특징을 사용
- JpaPagingItemReader에는 JPQL 문을 선언하고 EntityManagerFactory 전달

```
<bean id="itemReader" class="org.spr...JpaPagingItemReader">
  <property name="entityManagerFactory" ref="entityManagerFactory" />
  <property name="queryString" value="select c from CustomerCredit c" />
  <property name="pageSize" value="1000" />
</bean>
```

### ❑ Paging 기반 ItemReader③ - IbatisPagingItemReader

- 페이지의 Row를 읽을 수 있는 직접적인 지원은 하지 않으나 여러 표준화된 변수를 이용하여 쿼리 추가 가능

```
<bean id="itemReader" class="org.spr...IbatisPagingItemReader">
  <property name="sqlMapClient" ref="sqlMapClient" />
  <property name="queryId" value="getPagedCustomerCredits" />
  <property name="pageSize" value="1000" />
</bean>

<select id="getPagedCustomerCredits" resultMap="customerCreditResult">
  select id, name, credit from customer order by id
  asc LIMIT #_skiprows#,
  #_pagesize#
</select>
```



### ❑ Paging 기반 ItemReader ④ - MybatisPagingItemReader

- IbatisPagingItemReader와 설정 방식과 유사하나 설정 속성명, Query 작성법이 다른 차이가 있음  
(표준프레임워크에서 MybatisPagingItemReader 확장한 EgovMyBatisPagingItemReader를 제공)

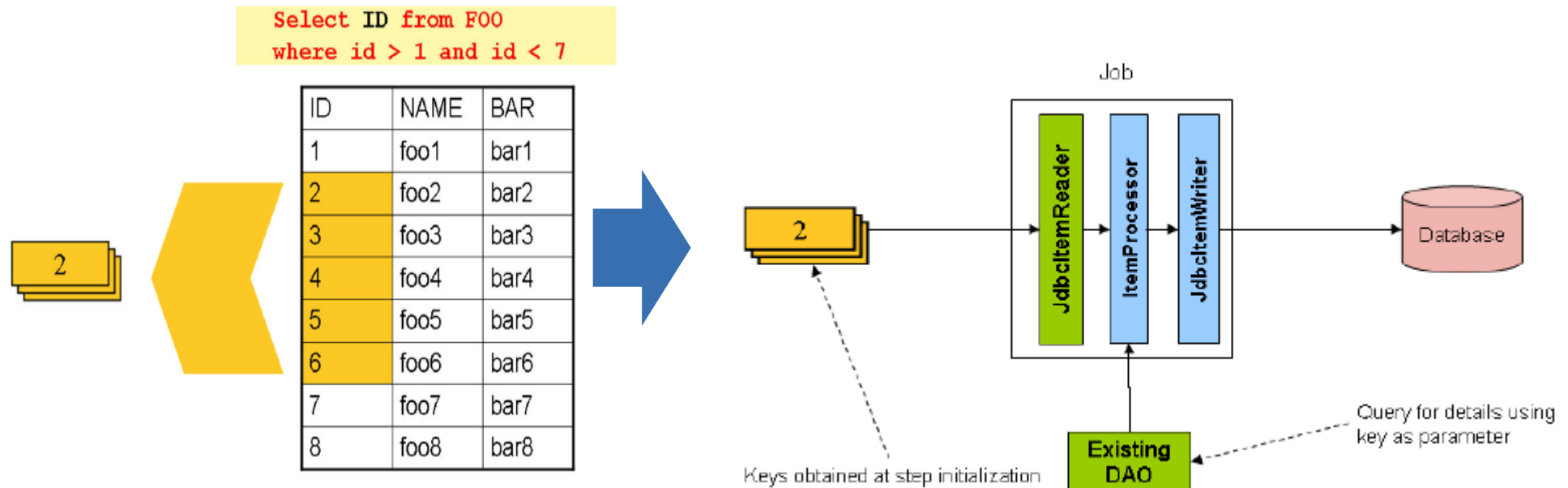
```
<bean id="itemReader" class="org.mybatis.spring.batch.MyBatisPagingItemReader" scope="step" >
  <property name="sqlSessionFactory" ref="sqlSession" />
  <property name="parameterValues" ref="stringParameters" />
  <property name="queryId" value="EmpMapper.selectEmpList" />
  <property name="pageSize" value="#{1000}" />
</bean>
```

## □ Driving Query 기반 처리

- 많은 애플리케이션 벤더들은 매우 극단적인 pessimistic Lock전략을 가지고 있음. (대표적인 경우 IBM, DB2)
- Pessimistic Lock전략(조회 시에도 Lock을 거는 전략)은 또 다른 온라인 애플리케이션에서도 테이블의 읽기가 필요한 경우 문제 원인이 될 수 있음.
- 극단적으로 양이 많은 데이터 집합에 대해서 Cursor를 여는 건 특정 벤더에 따라서 이슈가 될 수 있음.

## □ 동작 메커니즘

- 1단계 : 원하는 조회 조건으로 Primary Key를 우선 조회  
(단일키 : SingleColumnJdbcKeyCollector, 다수키 MultipleColumnJdbcKeyCollector)
- 2단계 : 조회한 Primary Key를 파라미터로 사용해서 처리(Processing)



### ❑ Spring Batch

- <http://static.springsource.org/spring-batch/2.1.x/reference/html/index.html>
- <http://static.springsource.org/spring-batch/3.0.x/reference/html/index.html>

---

# Q&A

감사합니다.