

SEНИМАР SPRING BATCH

THÀNH VIÊN:

NGUYỄN ĐỨC HUY - 3122410146

23 October 2025 →

- Introduce

- Basic

- Chiến lược xử lý theo lô (Batch Processing Strategies), Tổng quan kiến trúc xử lý theo lô (Batch Architecture Overview)

- Phân cấp công việc (Job Hierarchy), Chạy công việc (Running Job)

- Bước (Step), Xử lý theo khối (Chunk-oriented Processing), Tasklet

- Kiểm soát luồng bước (Controlling Step Flow)

- Bộ đọc dữ liệu (ItemReaders), Bộ xử lý dữ liệu (ItemProcessors), Bộ ghi dữ liệu (ItemWriters)

Next Slide →

INTRODUCTION

- Spring Batch là một framework xử lý theo lô (batch) nhẹ, toàn diện, được thiết kế để hỗ trợ phát triển các ứng dụng xử lý theo lô mạnh mẽ, đóng vai trò quan trọng trong các hoạt động hàng ngày của hệ thống doanh nghiệp.
- Spring Batch cung cấp các chức năng có thể tái sử dụng, cần thiết cho việc xử lý khối lượng lớn dữ liệu, bao gồm ghi log/theo dõi (logging/tracing), quản lý giao dịch (transaction management), thống kê quá trình xử lý công việc (job processing statistics), khởi động lại công việc (job restart), bỏ qua lỗi (skip), và quản lý tài nguyên (resource management).

USAGE SCENARIOS

Một chương trình xử lý theo lô (batch program) điển hình thường:

- Đọc một lượng lớn bản ghi từ cơ sở dữ liệu, tệp (file), hoặc hàng đợi (queue).
- Xử lý dữ liệu theo một cách nào đó.
- Ghi trả lại dữ liệu ở dạng đã được chỉnh sửa.

USAGE SCENARIOS

- Business Scenarios

- Cam kết (commit) quá trình xử lý theo lô định kỳ (Commit batch process periodically)
- Xử lý theo lô đồng thời (Concurrent batch processing): xử lý song song một công việc (job)
- Xử lý theo giai đoạn, dựa trên thông điệp trong hệ thống doanh nghiệp (Staged, enterprise message-driven processing)
- Xử lý theo lô song song quy mô lớn (Massively parallel batch processing)
- Khởi động lại thủ công hoặc theo lịch sau khi xảy ra lỗi (Manual or scheduled restart after failure)
- Xử lý tuần tự các bước phụ thuộc (Sequential processing of dependent steps) — có thể mở rộng thành xử lý theo luồng công việc (workflow-driven batches)
- Xử lý từng phần (Partial processing): bỏ qua các bản ghi (ví dụ, khi xảy ra rollback)
- Giao dịch toàn bộ lô (Whole-batch transaction): áp dụng cho các trường hợp có kích thước lô nhỏ hoặc sử dụng thủ tục lưu trữ/script có sẵn

BATCH PROCESSING STRATEGIES

1. Xử lý thông thường trong khung thời gian batch (Normal processing in a batch)

- Dữ liệu được cập nhật không cần thiết cho người dùng trực tuyến (on-line users) hoặc các quy trình batch khác, do đó không có vấn đề về đồng thời (concurrency), và có thể thực hiện một lần cam kết (commit) khi kết thúc toàn bộ quá trình batch.

2. Xử lý đồng thời giữa batch và trực tuyến (Concurrent batch or online)

- Dữ liệu có thể được người dùng trực tuyến cập nhật đồng thời không nên khóa bất kỳ dữ liệu nào (dù trong cơ sở dữ liệu hay trong tệp) mà người dùng trực tuyến có thể cần trong hơn vài giây.
- Ngoài ra, các bản cập nhật nên được cam kết (commit) vào cơ sở dữ liệu sau mỗi vài giao dịch, thay vì đợi đến khi kết thúc toàn bộ batch.

BATCH PROCESSING STRATEGIES

3. Xử lý song song (Parallel Processing)

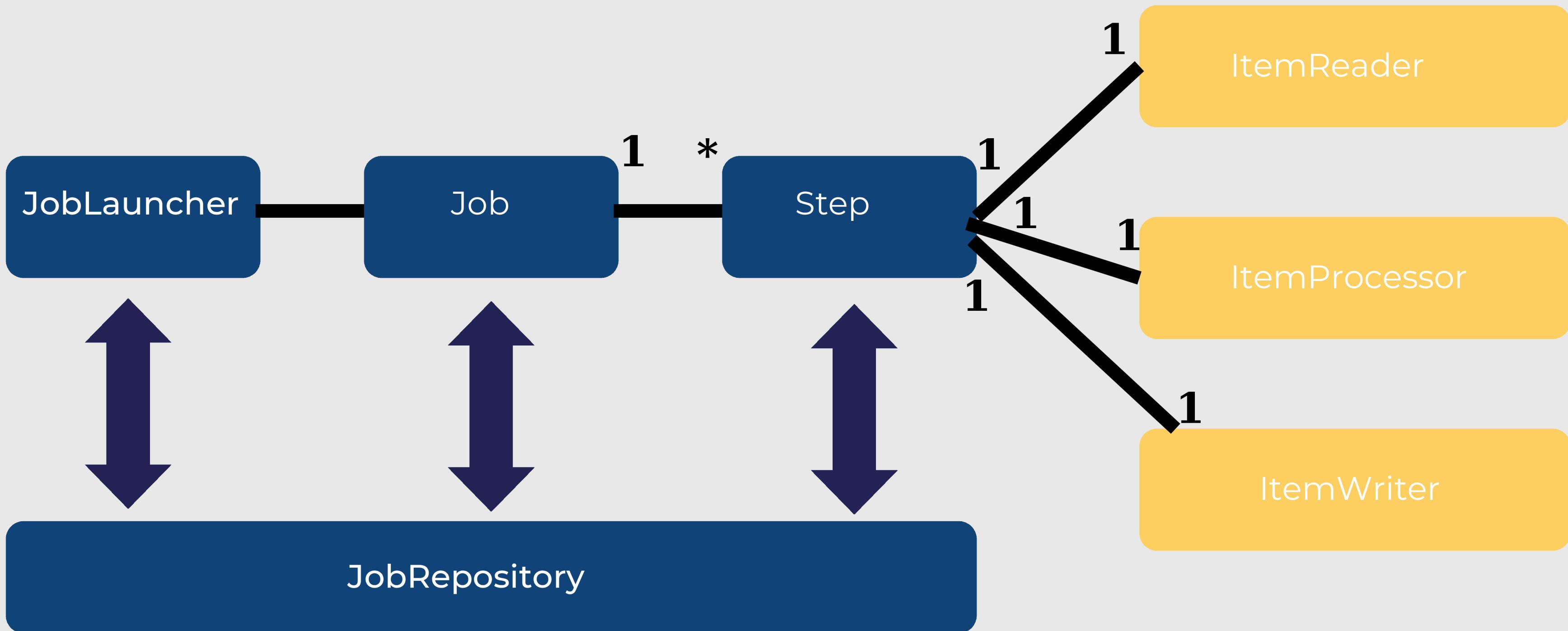
- Nhiều chạy batch hoặc job được thực thi song song nhằm giảm tổng thời gian xử lý batch.
- Các job này không chia sẻ cùng một tệp, bảng cơ sở dữ liệu hoặc không gian chỉ mục (index spaces).

4. Phân vùng (Partitioning)

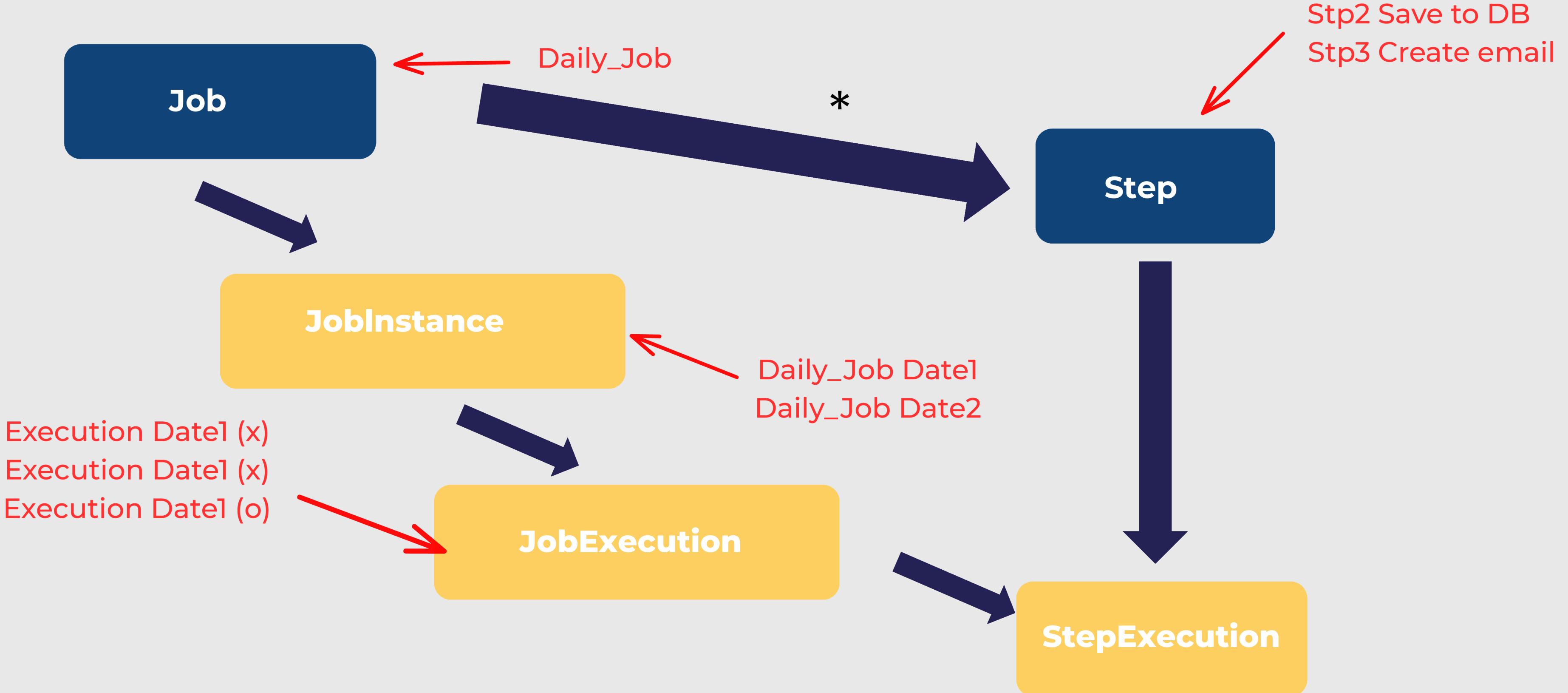
- Nhiều phiên bản của các ứng dụng batch lớn được chạy đồng thời để rút ngắn thời gian cần thiết cho việc xử lý các job batch kéo dài.
- Các quy trình có thể phân vùng thành công là những quy trình mà tệp đầu vào có thể được chia nhỏ và/hoặc các bảng chính trong cơ sở dữ liệu được phân vùng, cho phép ứng dụng chạy trên các tập dữ liệu khác nhau.



BATCH ARCHITECTURE OVERVIEW



JOB HIERARCHY



RUNNING JOB

- Launching a batch job requires two things:
 - Job
 - Job Launcher.
- Launching from the command line:
 - New JVM will be instantiated for each Job,
 - Every job will have its own JobLauncher.

implementation

```
<bash$> java CommandLineJobRunner io.spring.EndOfDayJobConfiguration endOfDay schedule.date(date)=2007/05/05
```

jobName

jobPath

jobParameters

RUNNING JOB

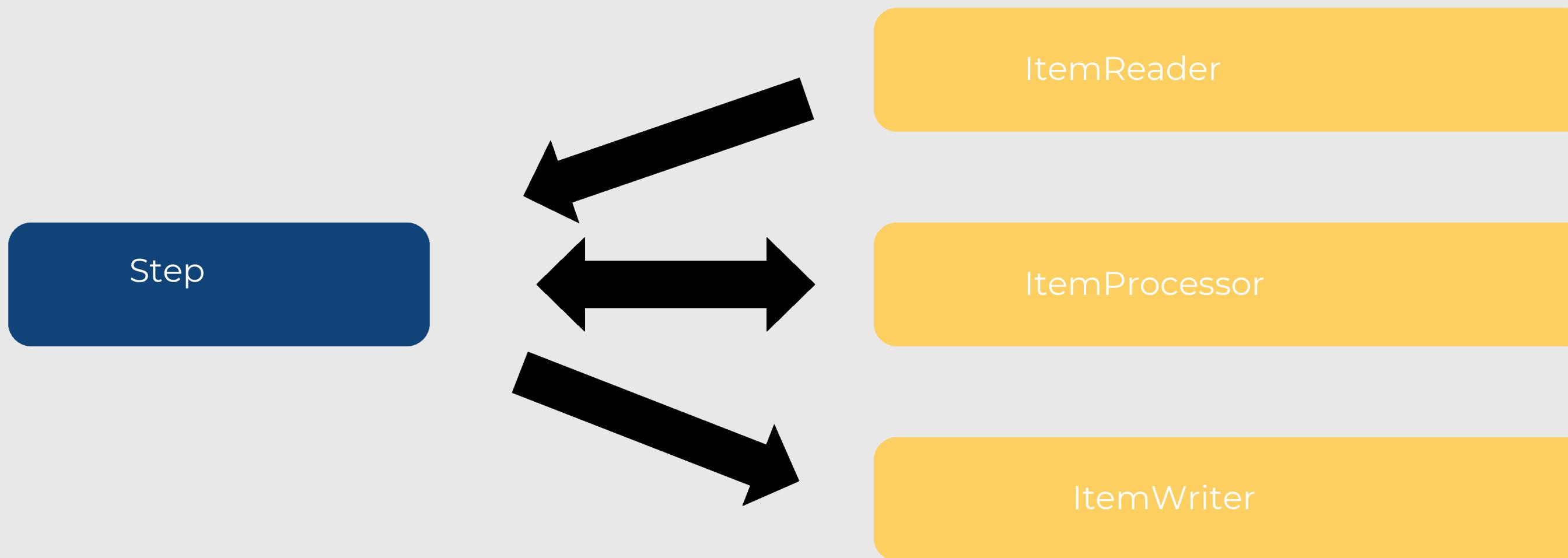
Khởi chạy từ trong web container (Launching from within a web container)

- Trong phạm vi của một HttpServletRequest (Within the scope of an HttpServletRequest)
- Một JobLauncher được cấu hình để khởi chạy công việc bất đồng bộ (asynchronous job launching)
- Nhiều yêu cầu (requests) sẽ được gọi để khởi chạy các job của riêng chúng.

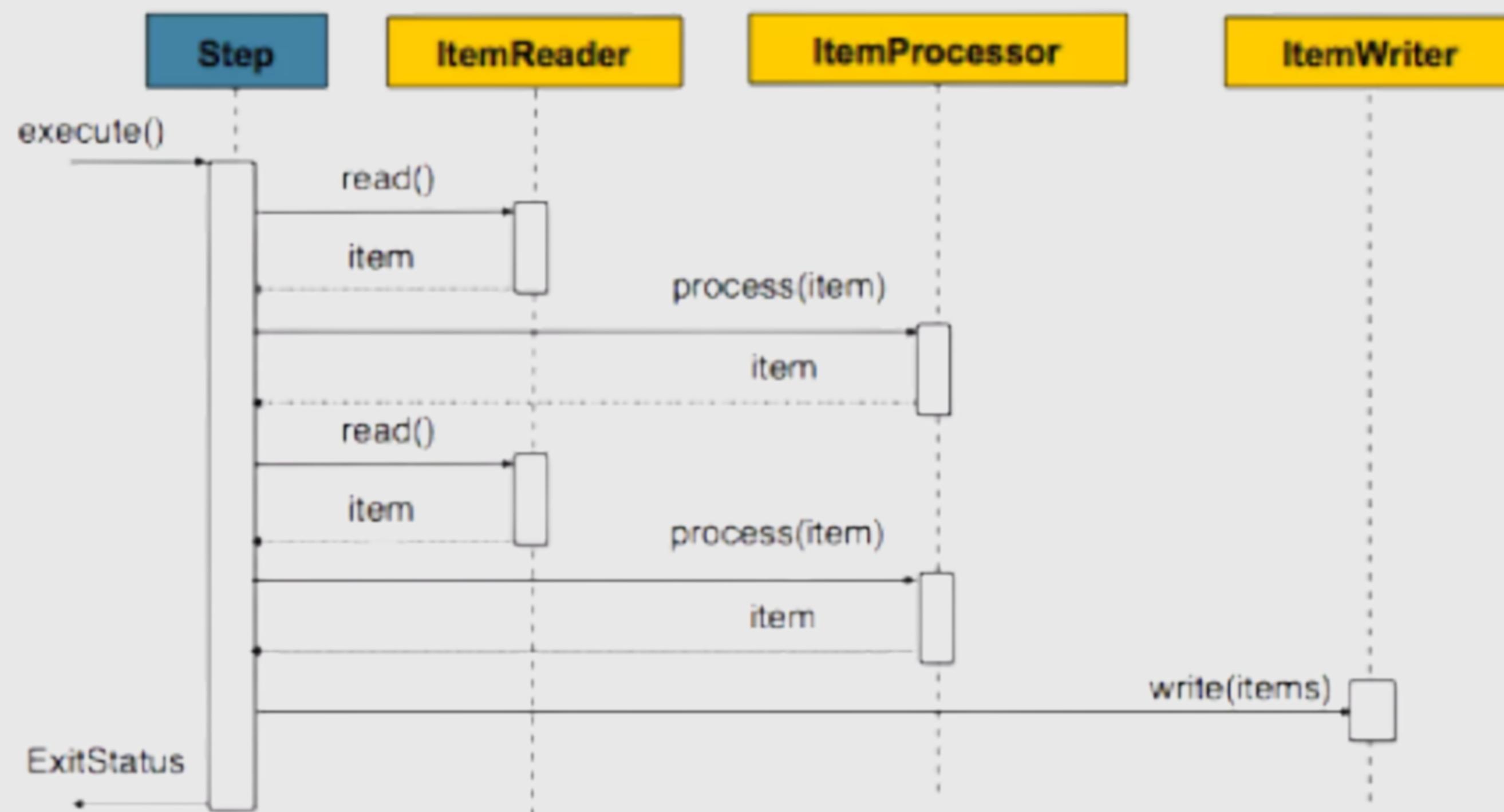
```
@Controller  
public class JobLauncherController {  
  
    @Autowired  
    JobLauncher jobLauncher;  
  
    @Autowired  
    Job job;  
  
    @RequestMapping("/jobLauncher.html")  
    public void handle() throws Exception {  
        jobLauncher.run(job, new JobParameters());  
    }  
}
```

STEP

- Step là một đối tượng miền (domain object) dùng để đóng gói một giai đoạn độc lập và tuần tự trong một công việc batch (batch job).
- Step chứa toàn bộ thông tin cần thiết để xác định và kiểm soát quá trình xử lý batch thực tế.



CHUNK-ORIENTED PROCESSING

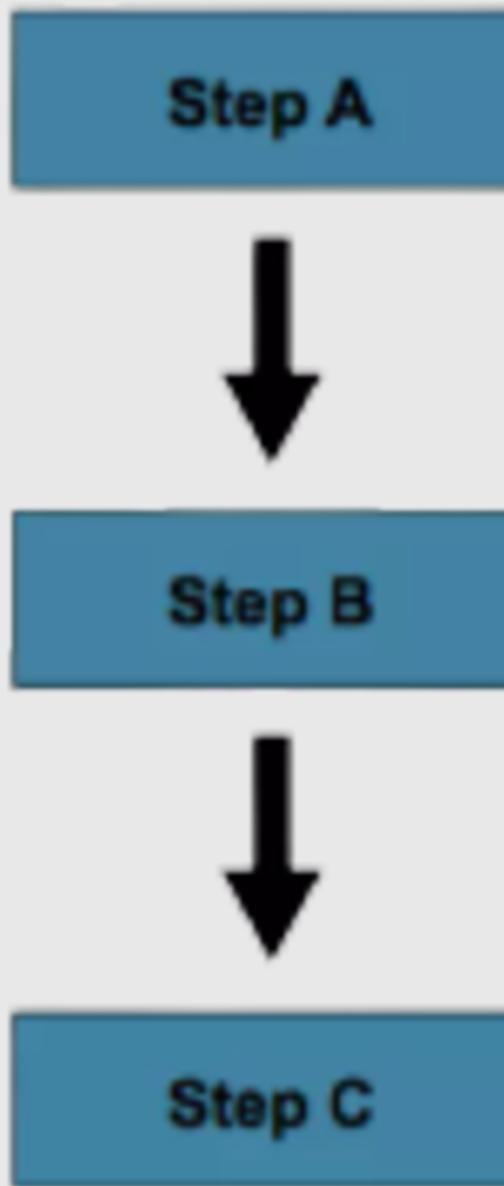


TASKLET

- Một giao diện đơn giản có một phương thức duy nhất là execute, được gọi lặp lại nhiều lần cho đến khi phương thức này trả về trạng thái FINISHED hoặc ném ra một ngoại lệ (exception) để báo hiệu lỗi.
- Những lớp triển khai Tasklet (Tasklet implementers) có thể gọi một thủ tục lưu trữ (stored procedure), một đoạn script, hoặc một câu lệnh SQL cập nhật đơn giản (simple SQL update statement).

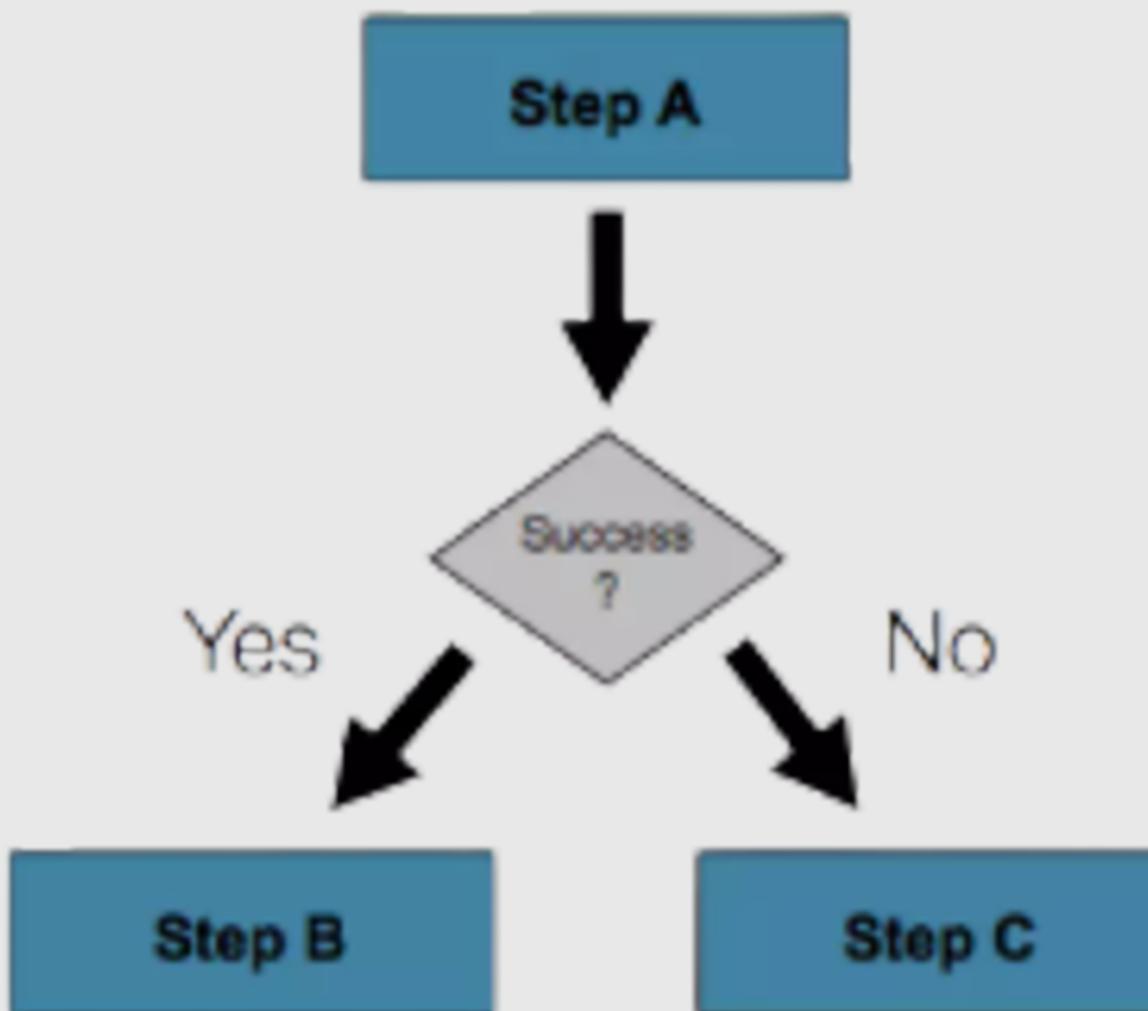
```
<job id="sampleJob" job-repository="jobRepository">
    <step id="step1" next="step2">
        <tasklet transaction-manager="transactionManager">
            <chunk reader="itemReader" writer="itemWriter" commit-interval="10"/>
        </tasklet>
    </step>
    <step id="step2">
        <tasklet ref="myTasklet"/>
    </step>
</job>
```

CONTROLLING STEP FLOW - SEQUENTIAL FLOW



```
@Bean  
public Job job() {  
    return this.jobBuilderFactory.get("job")  
        .start(stepA())  
        .next(stepB())  
        .next(stepC())  
        .build();  
}
```

CONTROLLING STEP FLOW - CONDITIONAL FLOW



```
@Bean  
public Job job() {  
    return this.jobBuilderFactory.get("job")  
        .start(stepA())  
        .on("*").to(stepB())  
        .from(stepA()).on("FAILED").to(stepC())  
        .end()  
        .build();  
}
```

ITEMREADERS

- Flat File: Các FlatFile ItemReaders đọc các dòng dữ liệu từ tệp phẳng (flat file) — loại tệp thường mô tả các bản ghi (records) với các trường dữ liệu (fields) được xác định bởi vị trí cố định trong tệp hoặc được phân tách bởi ký tự đặc biệt (ví dụ: dấu phẩy).
- XML: Các XML ItemReaders xử lý tệp XML một cách độc lập với các công nghệ được sử dụng để phân tích cú pháp (parsing), ánh xạ (mapping), và xác thực đối tượng (validating objects). Dữ liệu đầu vào cho phép xác thực tệp XML dựa trên lược đồ XSD (XSD schema).
- Database: Một nguồn dữ liệu từ cơ sở dữ liệu được truy cập để trả về các tập kết quả (result sets), sau đó có thể được ánh xạ thành các đối tượng để xử lý. Các triển khai mặc định của SQL ItemReader sử dụng RowMapper để trả về đối tượng, theo dõi hàng hiện tại nếu cần khởi động lại (restart), lưu trữ các thống kê cơ bản, và hỗ trợ cải thiện giao dịch (transaction enhancement).

ITEMREADERS

- DatabaseCursor

```
public JdbcCursorItemReader<CustomerCredit> itemReader() {  
    return new JdbcCursorItemReaderBuilder<CustomerCredit>()  
        .dataSource(this.dataSource)  
        .name("creditReader")  
        .sql("select ID, NAME, CREDIT from CUSTOMER")  
        .rowMapper(new CustomerCreditRowMapper())  
        .build();  
}
```

ITEMREADERS

- DatabasePaging

```
public JdbcPagingItemReader itemReader(DataSource dataSource, PagingQueryProvider queryProvider)
{
    Map<String, Object> parameterValues = new HashMap<>();
    parameterValues.put("status", "NEW");

    return new JdbcPagingItemReaderBuilder<CustomerCredit>()
        .name("creditReader")
        .dataSource(dataSource)
        .queryProvider(queryProvider)
        .parameterValues(parameterValues)
        .rowMapper(customerCreditMapper())
        .pageSize(1000)
        .build();
}
```

ITEMWRITERS

- ItemWriter có chức năng tương tự như ItemReader, nhưng thực hiện các thao tác ngược lại.
- Các tài nguyên (resources) vẫn cần được xác định vị trí, mở và đóng, tuy nhiên ItemWriter thực hiện ghi dữ liệu ra (write out) thay vì đọc dữ liệu vào (read in).
- Trong trường hợp cơ sở dữ liệu hoặc hàng đợi (queues), các thao tác này có thể là chèn (insert), cập nhật (update), hoặc gửi (send) dữ liệu.
- Định dạng tuần tự hóa (serialization format) của dữ liệu đầu ra phụ thuộc vào từng công việc batch (batch job) cụ thể.

ITEMPROCESSOR

- Cho một đối tượng đầu vào, tiến hành chuyển đổi (transform) nó và trả về một đối tượng khác.
- Đối tượng được cung cấp có thể cùng loại hoặc khác loại so với đối tượng đầu ra.
- Mục đích là cho phép áp dụng logic nghiệp vụ (business logic) trong quá trình xử lý, và việc xây dựng logic này hoàn toàn phụ thuộc vào lập trình viên.

```
public Step step1() {  
    return this.stepBuilderFactory.get("step1")  
        .<String, String>chunk(2)  
        .reader(fooReader())  
        .processor(fooProcessor())  
        .writer(barWriter())  
        .build();  
}
```

LOGGING ITEM PROCESSING AND FAILURES

```
public Step simpleStep() {
    return this.stepBuilderFactory.get("simpleStep")
        ...
        .listener(new ItemFailureLoggerListener())
        .build();
}
```

```
public class ItemFailureLoggerListener extends ItemListenerSupport {

    private static Log logger = LoggerFactory.getLog("item.error");

    public void onReadError(Exception ex) {
        logger.error("Encountered error on read", ex);
    }

    public void onWriteError(Exception ex, List<? extends Object> items) {
        logger.error("Encountered error on write", ex);
    }
}
```

ADVANTAGES OF SPRING BATCH

- Xử lý dữ liệu khối lượng lớn (Batch Processing):
- Hỗ trợ chia nhỏ, đọc – xử lý – ghi dữ liệu theo lô (chunk-oriented processing).
- Tích hợp tốt với hệ sinh thái Spring:
- Dễ cấu hình, tái sử dụng Spring Beans, Dependency Injection, Transaction Management.
- Quản lý trạng thái & retry:
- Theo dõi tiến trình Job, Step, restart lại từ nơi dừng, retry khi lỗi.
- Cấu trúc rõ ràng, tách biệt nhiệm vụ:
- Job → Step → Reader / Processor / Writer giúp dễ bảo trì, mở rộng.

ADVANTAGES OF SPRING BATCH

- Tích hợp scheduler dễ dàng:
- Kết hợp với Quartz, Spring Scheduler, hoặc cron job để chạy định kỳ.
- Khả năng mở rộng:
- Hỗ trợ multi-thread, partition, remote chunking cho xử lý song song.

DISADVANTAGES OF SPRING BATCH

- Độ phức tạp cao:
- Cấu hình nhiều, khó tiếp cận cho người mới.
- Không phù hợp cho xử lý thời gian thực:
- Thiết kế cho batch jobs (dữ liệu lớn, định kỳ), không cho request-response nhanh.
- Khó debug:
- Khi lỗi ở các bước (Step) phức tạp, log khó đọc, khó tái hiện.
- Cần cấu hình database lưu metadata:
 - Spring Batch dùng bảng metadata để lưu trạng thái Job → thêm bước cài đặt DB.
- Chi phí tài nguyên:
 - Nếu job lớn và nhiều step, tốn RAM/CPU đáng kể.

SCALING AND PARALLEL PROCESSING

Có hai chế độ xử lý song song (parallel processing):

- Một tiến trình, đa luồng (Single process, multi-threaded)
- Nhiều tiến trình (Multi-process)

Các chế độ này được chia nhỏ thành các loại sau:

- Bước đa luồng (Multi-threaded Step) – chạy trong một tiến trình duy nhất
- Các bước song song (Parallel Steps) – chạy trong một tiến trình duy nhất
- Xử lý theo khối từ xa (Remote Chunking of Step) – chạy trong nhiều tiến trình
- Phân vùng bước (Partitioning a Step) – có thể chạy trong một hoặc nhiều tiến trình

SPRING BATCH DEMO

THANK YOU

