Spring Batch Cheat Sheet

Spring Batch 6.0.x · Spring Framework 7 · Spring Boot 4.1.x · Java 17+ — finite, bounded bulk processing

DOMAIN MODEL

Type	Is
Job	definition: ordered Steps + config
JobInstance	Job + *identifying* JobParameters
JobExecution	one attempt at a JobInstance
Step	one independent phase
StepExecution	one attempt at a Step
JobRepository	persists all of the above
JobOperator	start/stop/restart/recover

Same identifying params → resume that instance. **Different** → new instance. A *completed* instance can never be rerun.

BatchStatus: STARTING → STARTED → COMPLETED | FAILED | STOPPING → STOPPED | ABANDONED | UNKNOWN

JOB SKELETON

```
new JobBuilder("endOfDayJob", repo)
  .incrementer(new RunIdIncrementer())
  .validator(new DefaultJobParametersValidator(
      new String[]{"run.date"},    // required
      new String[]{"output.dir"})) // optional
  .listener(jobListener)
  .start(s1).next(s2).build();
```

`.preventRestart()` · step-level `.startLimit(n)` · `.allowStartIfComplete(true)`

CHUNK LOOP

read 1 → process 1 → buffer → write N → commit. **chunk size = commit interval = transaction boundary = restart granularity.**

```
new StepBuilder("loadStep", repo)
  .<In, Out>chunk(500, tx)
  .reader(r).processor(p).writer(w)
  .stream(hiddenDelegate) // register ItemStream
  .build();               // delegates!
```

6.0 alternative: new `ChunkOrientedStepBuilder<In,Out>(name, repo, 500)`. Dynamic size: `.chunk(completionPolicy, tx)`.

TASKLET STEP

```
new StepBuilder("archiveStep", repo)
  .tasklet((contribution, chunkContext) -> {
      service.doOneThing();
      return RepeatStatus.FINISHED; // or
CONTINUABLE
  }, tx).build();
```

Adapters: MethodInvoking / Callable / SystemCommand. One transaction, all-or-nothing. For DDL, file moves, one remote call, stored procs.

FAULT TOLERANCE

```
.faultTolerant()
  .skip(FlatFileParseException.class)
  .skip(ValidationException.class)
  .noSkip(FileNotFoundException.class)
  .skipLimit(50)
  .retry(DeadlockLoserDataAccessException.class)
  .retryLimit(3)
  .backOffPolicy(exponentialBackOff())
  .noRollback(BusinessWarning.class)
  .listener(skipListener)
```

Replay: a failed chunk rolls back, is re-processed and re-written item-by-item → **the ItemProcessor must be idempotent**; side effects belong in the writer. Escape hatch: `.processorNonTransactional()`. Retry = Spring Framework 7 `core.retry`, not Spring Retry.

STEP FLOW & LISTENERS

```
.start(loadStep)
  .on("FAILED").to(alertStep)
.from(loadStep)
  .on("COMPLETED WITH SKIPS").to(fixStep)
.from(loadStep).on("*").to(reportStep)
.next(decider)
  .on("WEEKEND").to(shortReport)
.end()      // .fail() | .stopAndRestart(step)
```

Wildcards * / ?. Parallel: `.split(taskExecutor).add(flow)`. Listeners: StepExecution, Chunk, ItemRead/Process/Write, Skip, Retry, JobExecution (`@BeforeStep`, `@AfterJob`…).

ITEMREADERS

Source	Reader
CSV / fixed	FlatFileItemReader
many files	MultiResourceItemReader
XML	StaxEventItemReader
JSON	JsonItemReader
JDBC	JdbcCursor / JdbcPaging
JPA / HB	JpaCursor|Paging, HibernateCursor|Paging
proc	StoredProcedureItemReader
Spring Data	RepositoryItemReader, MongoPaging

Cursor: cheap/row, one long tx, single-thread. **Paging**: restartable, thread-safe, needs a unique sort key.

```
new FlatFileItemReaderBuilder<T>().name("r")
  .resource(res).linesToSkip(1).delimited()
  .names("a","b").targetType(T.class).build();
```

ITEMPROCESSORS & WRITERS

`process()` returns `null` → **filter** (filterCount). Throw → **skip** (processSkipCount).

Compose: CompositeItemProcessor (chain) · ClassifierComposite (route) · ItemProcessorAdapter · Bean/ValidatingItemProcessor.

```
new JdbcBatchItemWriterBuilder<T>
().dataSource(ds)
  .sql("INSERT INTO t (a,b) VALUES (:a,:b)")
  .beanMapped()          // or .columnMapped()
  .assertUpdates(false) // false for upserts
  .build();
```

Others: FlatFile (LineAggregator + FieldExtractor, header/footer), Stax/JsonFile, MultiResource, Jpa/Hibernate, Repository/Mongo, ItemWriterAdapter, Composite/Classifier.

METADATA SCHEMA

- BATCH_JOB_INSTANCE — JOB_NAME + JOB_KEY
- BATCH_JOB_EXECUTION — STATUS, EXIT_CODE
- BATCH_JOB_EXECUTION_PARAMS
- BATCH_JOB_EXECUTION_CONTEXT
- BATCH_STEP_EXECUTION — the counters
- BATCH_STEP_EXECUTION_CONTEXT

Sequences: BATCH_JOB_SEQ, BATCH_JOB_EXECUTION_SEQ, BATCH_STEP_EXECUTION_SEQ. DDL `org/springframework/batch/core/schema-*.sql`; create at ISOLATION_SERIALIZABLE.

Counters: readCount = writeCount + filterCount + processSkipCount + writeSkipCount

PROPERTIES & LAUNCHING

```
spring.batch.job.name=endOfDayJob
spring.batch.job.enabled=true
spring.batch.jdbc.initialize-schema=never
spring.batch.jdbc.table-prefix=BATCH_
spring.batch.jdbc.platform=postgresql

java -jar app.jar \
  --spring.batch.job.name=endOfDayJob \
  run.date=2026-01-31
chunk.size=500,java.lang.Long
```

JobOperator: `start` · `startNextInstance` · `restart` · `stop` · `abandon` · `recover`. CLI: **CommandLineJobOperator** (replaces CommandLineJobRunner).

INFRASTRUCTURE — 6.0

Do NOT add `@EnableBatchProcessing` under Spring Boot — it disables the batch auto-configuration.

Manual: `@EnableBatchProcessing` + `@EnableJdbcJobRepository` / `@EnableMongoJobRepository`, or extend `DefaultBatchConfiguration` (replaces `BatchConfigurer`).

Defaults are **resourceless**: ResourcelessJobRepository / TransactionManager — no DB (so no restart or history).

Unified: JobRepository *extends* JobExplorer · JobOperator *extends* JobLauncher · JobRegistry optional. Gone: JobBuilderFactory, StepBuilderFactory, javax.*, JUnit 4.

SCALING — MEASURE FIRST

Option	Restart
Multi-threaded step (`.taskExecutor`)	poor
Parallel steps (`.split`)	good
Async processor/writer	as step
Local partitioning	excellent
Remote partitioning	excellent
Remote chunking	moderate

```
new StepBuilder("manager", repo)
  .partitioner("workerStep", partitioner)
  .partitionHandler(taskExecutorPartitionHandler)
  .build();
```

Partitioner → StepExecutionSplitter → PartitionHandler (gridSize) → N worker StepExecutions over disjoint slices. Multi-threaded steps need SynchronizedItemStreamReader.

LATE BINDING (SPEL)

```
@Bean @StepScope
Reader r(@Value("#{jobParameters['run.date']}")
LocalDate d,
         @Value("#{jobExecutionContext['file']}")
String f,
         @Value("#
{stepExecutionContext['minId']}") Long id)
```

Roots: `jobParameters`, `jobExecutionContext`, `stepExecutionContext`, `stepExecution/jobExecution`. Pass data between steps with `ExecutionContextPromotionListener.setKeys(...)`.

OBSERVABILITY

Meter	Type
spring.batch.job	Timer
spring.batch.job.active	LongTaskTimer
spring.batch.step	Timer
spring.batch.item.read	Timer
spring.batch.item.process	Timer
spring.batch.chunk.write	Timer

One Observation per job / step / chunk → spans with a tracer. 6.0 adds **JFR events**: -XX:StartFlightRecording=filename=b.jfr.

TESTING

JUnit Jupiter only — JUnit 4 support removed in 6.0.

```
@SpringBatchTest
@SpringJUnitConfig(BatchConfig.class)
class JobTests {
  @Autowired JobOperatorTestUtils utils;
  @Autowired JobRepositoryTestUtils repoUtils;
  @AfterEach void clean() {
    repoUtils.removeJobExecutions(); }
  @Test void runs() throws Exception {
    JobExecution e = utils.startJob(params);
    assertEquals(COMPLETED, e.getStatus()); }
  // one step: utils.startStep("loadStep",
params, ctx);
}
```

`MetaDataInstanceFactory` builds domain objects with no repository. Step-scoped beans: a `getStepExecution()` method, or `StepScopeTestUtils.doInStepScope(...)`.

TUNING & ANTI-PATTERNS

- Chunk 100–1000; never 1, never 100k.
- Set `fetchSize`; page on an indexed unique key.
- `saveState(false)` when not restartable.
- Keep the ExecutionContext tiny (written every commit).
- Index & prune the metadata tables.
- Avoid: per-item transactions, chatty readers (N+1), side effects in processors, parallelising before measuring.