Spring Batch Essentials

Spring Batch is an open source, lightweight, and comprehensive solution designed to enable the development of robust batch applications that are vital for enterprise operations.

Packed with real-world examples, this book starts with an insight into the batch applications and Spring Batch offerings. After exploring the architecture and key components, you will learn how to develop and execute a batch application. While gaining insights on the essential configurations and execution techniques for batch jobs, you will learn about the key technical implementations of the read, write, and processing features for different forms of data. Next, you will move on to the key features such as transaction management, job flows, job monitoring, and data sharing across the steps of the executing jobs. Finally, you will learn how Spring Batch can integrate with diverse enterprise technologies and facilitate optimization and performance improvement with scaling and partitioning techniques.

Who this book is written for

If you are a Java developer with basic knowledge of Spring and some experience in the development of enterprise applications, and want to learn about batch application development in detail, then this book is ideal for you. This book will be perfect as your next step towards building simple yet powerful batch applications on a Java-based platform.

What you will learn from this book

- Understand the infrastructure to design, develop, and execute a batch application
- Configure batch jobs using XML, Expression Language and annotations, and execute from diverse platforms
- Develop batch jobs with the essential read, process, and write features different forms of data
- Get to grips with techniques you can use to manage transactions, control job flows, and utilize data sharing
- Integrate Spring Batch with other technologies to develop robust batch applications on an enterprise platform
- Monitor and report the batch job execution updates by accessing live job execution information
- Optimize scale and performance improvement with parallel processing techniques
- Perform unit integration and functional testing on Spring Batch applications

Community Experience Distilled

Design, develop, and deliver robust batch applications with the power of the Spring Batch framework

P. Raja Malleswara Rao

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Spring Batch Fundamentals'
- A synopsis of the book’s content
- More information on Spring Batch Essentials

About the Author

P. Raja Malleswara Rao is a senior consultant, focusing on enterprise architecture and development of Java-related technologies. He is a certified Java and web components developer with deep expertise in building enterprise applications using diverse frameworks and methodologies. He is an active participant in technical forums, groups, and conferences. He has worked with several Fortune 500 organizations and is passionate about learning new technologies and their developments.

I would like to thank my family and friends for their love and support, especially my wife, Bhargavi, for sacrificing the quality time I could have spent with her and encouraging me through this stint.

I would also like to thank my reviewers for their valuable suggestions in improving the quality of this book, my colleagues, and the Spring community for sharing great thoughts that helped me a lot in keeping myself updated.
Welcome to the world of Spring Batch! We're pleased that you have chosen our book, fully dedicated to the essentials of Spring Batch 3.0.2 release. Before we get started with this book, I would like to give an overview of how this book is organized and how you can get the most out of it. Once you have completed reading this book, you should be familiar with key batch processing concepts and understand how to solve the majority of the real-world problems that you will need to solve with Spring Batch. This book gives an insight of the essential concepts and applications of the Spring Batch framework, which will allow you to handle any unexpected use cases the book does not cover.

Spring Batch is an open source, lightweight, and comprehensive solution, designed to enable the development of robust batch applications, vital for enterprise operations.

This tutorial starts with an insight into batch applications and Spring Batch offerings. Understanding the architecture and key components, you will learn how to develop and execute a batch application. You can then walk through the essential configurations and execution techniques, key technical implementations of the read and write, and processing features of different forms of data. Next, you will move onto the key features, such as transaction management, job flows, job monitoring, and data sharing across the steps of the executing jobs. Finally, you will learn how Spring Batch can integrate with diverse enterprise technologies and facilitate optimization and performance improvement with scaling and partitioning techniques.

This book uses a simple application based on Spring Batch to illustrate how to solve real-world problems. The application is intended to be very simple and straightforward, and purposely contains very little functionality—the goal of this application is to encourage you to focus on the Spring Batch concepts, and not get tied up in the complexities of application development. You will have a much easier time following the book if you take the time to review the sample application source code. Some tips on getting started are found in the appendix.
What This Book Covers

Chapter 1, *Spring Batch Fundamentals*, introduces you to batch applications and details about Spring Batch and its offerings. It discusses Spring Batch architecture, and also demonstrates how to design and execute a Spring Batch job.

Chapter 2, *Getting Started with Spring Batch Jobs*, covers the Spring Batch XML features, how to configure jobs, transactions and repository, working with EL and listeners, and executing jobs from command line and web application schedulers.

Chapter 3, *Working with Data*, demonstrates the essential data handling mechanisms, including reading the data from different sources such as flat files, XML and databases, processing the data, and writing the data to different destinations. It also explains about transforming and validating the data in the processing data phase.

Chapter 4, *Handling Job Transactions*, covers transactions and Spring Batch transaction management, and explains how to customize the transaction along with transaction patterns.

Chapter 5, *Step Execution*, explains the techniques to control the job flow, sharing the data between steps in execution and process reuse with externalization. It also demonstrates terminating the batch job in different states and their importance.

Chapter 6, *Integrating Spring Batch*, covers the integration techniques for enterprise applications based on Spring Batch, and some of the job launching and process techniques.

Chapter 7, *Inspecting Spring Batch Jobs*, demonstrates the monitoring techniques of Spring Batch jobs, access methods to execute job data, monitoring and reporting with listeners and web monitoring techniques.

Chapter 8, *Scaling with Spring Batch*, discusses the performance and scaling concerns with Spring Batch support for configuration-based tuning techniques such as parallel processing, remote chunking, and partitioning.

Chapter 9, *Testing the Spring Batch*, covers details of testing techniques for Spring Batch applications with the help of open source frameworks and Spring support for unit testing, integration testing, and functional testing.

Appendix covers the setup references for Java, Eclipse IDE, a project with dependencies, and an administration project.
Organizations need to process huge volumes of data through a series of transactions in their day-to-day operations. These business operations should be automated to process the information efficiently without human intervention. Batch processing can execute such a series of operations through programs, with a predefined set of data groups as input, process the data, and generate a set of output data groups and/or update the database.

In this chapter, we will cover the following topics:

- Introduction to batch applications
- Spring Batch and its offerings
- Spring Batch infrastructure
- Job design and executions

**Introduction to batch applications**

Organizations need to accomplish diverse business operations that include a large amount of data processing. Following are some examples of such operations:

- Generation of salary slips and tax calculations in a large enterprise
- Credit card bill generation by banks
- Fresh stock updated by retail stores in their catalog
All such operations are executed with a predefined set of configurations and schedules, to run at a particular offload system time. Batch applications should be able to process large volumes of data without human intervention. The following figure represents a typical batch application:

A standard batch application is expected to have the following capabilities:

- **Scalable**: It should be able to process billions of records and be reliable without crashing the application
- **Robust**: It should be intelligent enough to identify the invalid data and keep track of such mishaps to rerun with corrected data
- **Dynamic**: It should interact with different systems to access the data using the credentials provided and process the operations
- **Concurrent**: It must process multiple jobs in parallel with the shared resources
- **Systematic**: It should process the workflow-driven batches in a sequence of dependent steps
- **High performance**: It must complete the processing in a specified batch window
Spring Batch and its offerings

Spring Batch is a lightweight, comprehensive batch framework designed to enable the development of robust batch applications that are vital for the daily operations of enterprise systems developed by SpringSource and Accenture in collaboration.

Spring Batch follows POJO-based development to let developers easily implement batch processing and integrate with other enterprise systems when needed.

Plain Old Java Object (POJO) represents an ordinary Java object that can be used to store a data item and exchange information between services easily.

While Spring Batch provides many reusable functions adopted from the Spring framework and customized for batch applications to perform common batch (such as split processing of huge volumes of data, logging, transaction management, job process-skip-restart, and effective resource management), it is not a scheduling framework. Spring Batch can work in conjunction with a scheduler (such as Quartz/Control-M), but cannot replace a scheduler.

We discussed the capabilities expected from a standard batch application in the previous section. Spring Batch is designed to fulfill the expected features, along with its high capability, to integrate with different applications developed in other frameworks. Let’s observe some of the important features offered by Spring Batch:

- Support for multiple file formats, including fixed length, delimited files, XML and common database access using JDBC, and other prominent frameworks
- Automatic retry after failure
- Job control language to monitor and perform common operations such as job start, stop, suspend, and cancel
- Tracking status and statistics during the batch execution and after completing the batch processing
- Support for multiple ways of launching the batch job, including script, HTTP, and message
- Support to run concurrent jobs
- Support for services such as logging, resource management, skip, and restarting the processing
Spring Batch infrastructure
Spring Batch is designed with a layered architecture, including three major components, namely, Application, Core, and Infrastructure, as shown in the following figure:

The Application layer contains the developer-written code to run the batch jobs using Spring Batch.

The Batch Core layer contains the core runtime classes such as JobLauncher, Job, and Step, necessary to launch and control the batch job. This layer interacts with the Application layer and Batch Infrastructure layer to run the batch jobs.

The Batch Infrastructure layer contains the common readers, writers, and services. Both Application and Batch Core are built on top of Infrastructure. They refer to Infrastructure for the information required to run the batch jobs.

Multiple components are involved in Spring Batch job execution. The components and their relationship are discussed in the next section.

Spring Batch components
The following figure represents the Spring Batch job components and the relationship between these components:
JobLauncher is the interface responsible for beginning a job. When a job is first launched, JobLauncher verifies in the JobRepository, if the job is already executed and the validity of the Job parameter before executing the job.

A job is the actual batch process to be executed. A Job parameter can be configured in an XML or a Java program.

JobInstance is the logical instance of the job per cycle. If a JobInstance execution fails, the same JobInstance can be executed again. Hence, each JobInstance can have multiple job executions.

JobExecution is the representation of single run of a job. JobExecution contains the run information of the job in execution, such as status, startTime, endTime, failureExceptions, and so on.

JobParameters are the set of parameters used for a batch job.

A Step is a sequential phase of a batch job. Step contains the definition and control information of a batch job. The following figure represents multiple steps in a batch job. Each Step constitutes three activities, namely, data reading, processing, and writing, which are taken care of by ItemReader, ItemProcessor, and ItemWriter respectively. Each record is read, processed (optional), and written to the system.

StepExecution is the representation of a single run of a Step. StepExecution contains the run information of the step, such as status, startTime, endTime, readCount, writeCount, commitCount, and so on.

JobRepository provides create, retrieve, update, and delete (CRUD) operations for the JobLauncher, Job, and Step implementations.

ItemReader is the abstract representation of the retrieval operation of Step. ItemReader reads one item at a time.
ItemProcessor is the abstract representation of the business processing of the item read by ItemReader. ItemProcessor processes valid items only and returns null if the item is invalid.

ItemWriter is the abstract representation of the output operation of Step. ItemWriter writes one batch or chunk of items at a time.

In the next section, we will use our understanding of these components and develop a simple batch application using the essential Spring Batch job components. Also included are the code snippets of this application in steps.

Job design and executions

Spring Batch can be configured in your project in multiple ways, by including downloaded ZIP distribution and checking out from Git or configure using Maven. In our case, we will use the Maven configuration. You should have Maven installed in your system directly or using an IDE-based plugin (we are using Eclipse in this example). Refer to https://www.eclipse.org/m2e/ to integrate Maven in your Eclipse IDE. The latest versions of Eclipse come with this plugin installed; verify this before installing.

A Spring Batch job can be launched in multiple ways, including the following:

- Launching the job from the command line
- Launching the job using job schedulers
- Launching the job from a Java program
- Launching the job from a web application

For this sample program, we are launching the batch job from a simple Java program.

The following are the steps, with code snippets, to run the first batch job using Spring Batch:

1. Create a Maven-enabled Java project (let's call it SpringBatch). Maven is the software to manage the projects effectively. The pom.xml file is the configuration file for Maven to include any API dependencies. There are dedicated Maven archetypes that can create sample projects. The location for Maven is http://mvnrepository.com/artifact/org.springframework.batch/spring-batch-archetypes.

2. Configure pom.xml in the root directory of your project to have the required Maven dependencies that include the following:
   - Spring framework with batch
- **log4j** for logging
- **JUnit** to test the application
- Commons Lang helper utilities for the `java.lang` API
- **HyperSQL Database (HSQLDB)** to be able to run using HSQLDB, which is a relational database management system written in Java

```xml
  <modelVersion>4.0.0</modelVersion>
  <groupId>batch</groupId>
  <artifactId>SpringBatch</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <properties>
    <spring.framework.version>3.2.1.RELEASE</spring.framework.version>
    <spring.batch.version>3.0.2.RELEASE</spring.batch.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>commons-lang</groupId>
      <artifactId>commons-lang</artifactId>
      <version>2.6</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.batch</groupId>
      <artifactId>spring-batch-core</artifactId>
      <version>${spring.batch.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.batch</groupId>
      <artifactId>spring-batch-infrastructure</artifactId>
      <version>${spring.batch.version}</version>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.17</version>
    </dependency>
  </dependencies>
</project>
```
3. Create `log4j.xml` under the `src\main\resources` directory to log with the following content, which will produce a formatted console output:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
  <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.out" />
    <param name="Threshold" value="INFO" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%d %-5p %c - %m%n"/>
    </layout>
  </appender>
  <logger name="org.springframework" additivity="false">
    <level value="INFO"/>
    <appender-ref ref="CONSOLE"/>
  </logger>
</log4j:configuration>
```
4. Include the configuration file (context.xml) under the src\main\resources\batch directory with the following content. Context configuration includes the jobRepository, jobLauncher, and transactionManager configuration. We configured the batch as the default schema in this configuration.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/batch"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <beans:bean id="jobRepository"
    class="org.springframework.batch.core.repository.MapJobRepositoryFactoryBean">
    <beans:property name="transactionManager"
      ref="transactionManager"/>
  </beans:bean>
  <beans:bean id="jobLauncher"
    class="org.springframework.batch.core.launch.support.SimpleJobLauncher">
    <beans:property name="jobRepository"
      ref="jobRepository"/>
  </beans:bean>
  <beans:bean id="transactionManager"
    class="org.springframework.batch.support.transaction.ResourcelessTransactionManager"/>
</beans:beans>
```

5. Include the job config (firstBatch.xml) under the src\main\resources\batch directory with the following content. Batch job configuration includes configuring the batch job with step and tasklet, using a Java program.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/batch"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <beans:bean id="jobRepository" class="org.springframework.batch.core.repository.MapJobRepositoryFactoryBean">
    <beans:property name="transactionManager" ref="transactionManager"/>
  </beans:bean>
  <beans:bean id="jobLauncher" class="org.springframework.batch.core.launch.SimpleJobLauncher">
    <beans:property name="jobRepository" ref="jobRepository"/>
  </beans:bean>
  <beans:bean id="transactionManager" class="org.springframework.batch.support.transaction.ResourcelessTransactionManager"/>
</beans:beans>
```
http://www.springframework.org/schema/batch
http://www.springframework.org/schema/batch/spring-batch-3.0.xsd">
  <beans:import resource="context.xml" />
  <beans:bean id="firstBatch" class="batch.FirstBatch"/>
  <step id="firstBatchStepOne">
    <tasklet ref="firstBatch"/>
  </step>
  <job id="firstBatchJob">
    <step id="stepOne" parent="firstBatchStepOne"/>
  </job>
</beans:beans>

6. Write the tasklet (the strategy for processing in a step) for the first job (FirstBatch.java) under the src\main\java\batch directory with the following content. This tasklet program is referred to in the firstBatch.xml configuration for tasklet reference under Job.

```java
package batch;
import org.apache.log4j.Logger;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.repeat.RepeatStatus;
public class FirstBatch implements Tasklet {
    static Logger logger = Logger.getLogger("FirstBatch");

    public RepeatStatus execute(StepContribution arg0,
            ChunkContext arg1)
        throws Exception {
        logger.info("** First Batch Job is Executing! **");
        return RepeatStatus.FINISHED;
    }
}
```

7. Write the Java program to execute the batch job (ExecuteBatchJob.java) under the src\main\java\batch directory with the following content. Through this program, we access the job configuration file and identify the JobLauncher and Job beans from the configuration files. JobExecution is invoked from the run method of JobLauncher by passing the job and jobParameters.
As mentioned earlier, we can run a batch job from either of the options, including command line, job schedulers, web application, or a simple Java program. We are using a simple Java program here to run our first job.

```java
package batch;

import org.apache.log4j.Logger;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ExecuteBatchJob {

    static Logger logger =
            Logger.getLogger("ExecuteBatchJob");
    public static void main(String[] args) {

        String[] springConfig = {"batch/firstBatch.xml"};
        ApplicationContext context = new
                ClassPathXmlApplicationContext(springConfig);

        JobLauncher jobLauncher = (JobLauncher)
                context.getBean("jobLauncher");
        Job job = (Job) context.getBean("firstBatchJob");
        try {
            JobExecution execution = jobLauncher.run(job, new
                    JobParameters());
            logger.info("Exit Status : " +
                    execution.getStatus());
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (context != null) {
                context = null;
            }
        }
        logger.info("Done");
    }
}
```
8. Following is the folder structure to be generated in the SpringBatch project, after including the resources mentioned earlier:

```
SpringBatch
  .settings
  bin
  src
    main
      java
        batch
          ExecuteBatchJob.java
          FirstBatch.java
      resources
        batch
          context.xml
          firstBatch.xml
          log4j.xml
    target
      classespath
      project
      pom.xml
```

Add src/main/java and src/main/resources to the project source through build path properties, as shown in the following screenshot:
9. Build the project with the Maven installation and run the `ExecuteBatchJob` Java program to get the batch job execution status printed on the console:

```
2014-06-01 17:02:29,548 INFO  org.springframework.batch.core.launch.support.SimpleJobLauncher - Job: [FlowJob: [name=firstBatchJob]] launched with the following parameters: [{}]
2014-06-01 17:02:29,599 INFO ** First Batch Job is Executing! **
2014-06-01 17:02:29,633 INFO  org.springframework.batch.core.launch.support.SimpleJobLauncher - Job: [FlowJob: [name=firstBatchJob]] completed with the following parameters: [{}]) and the following status: [COMPLETED]
2014-06-01 17:02:29,637 INFO Exit Status :COMPLETED
2014-06-01 17:02:29,639 INFO Done
```

Following the previously mentioned steps, we configured our first batch job using Spring Batch and executed it successfully from a Java program.

**Summary**

Throughout this chapter, we learned about batch applications, real-time batch applications, and the capabilities expected from a standard batch application. We also learned about Spring Batch applications and the features offered by the Spring Batch technology, high-level Spring Batch architecture, and components involved in Spring Batch job execution, along with the relationships among those components. We completed this chapter with the development of a simple batch application and ran the program successfully.

In the next chapter, we will learn about the configuration of batch jobs using XML and EL, and the execution of batch jobs from the command line and application. We will also discuss the scheduling of batch jobs.
Where to buy this book
You can buy Spring Batch Essentials from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.