Spring Integration Essentials

Integrate the heterogeneous endpoints of enterprise applications with Spring Integration for effective communication

Chandan Pandey
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 "Getting Started"
- A synopsis of the book’s content
- More information on Spring Integration Essentials

About the Author

Chandan Pandey is an Oracle Certified Enterprise architect with more than 10 years of experience in designing, coding, and delivering enterprise applications using various tools and technologies. He has always been amused by the power of software that reduces the boredom of repetitive tasks and introduces agility and efficiency. He firmly believes that tools, technology, and methodology are a medium to provide a solution and would like to be positioned as a solutions expert rather than limiting his identity to a framework or tool. This is reflected in the breadth and depth of his work; he is proficient not only in traditional languages and frameworks such as Java/JEE and Spring, but also in Groovy and Grails, Vaadin, and Play Framework, to name a few. His domain experience varies from blogging applications, web frameworks, content management systems, and finance, to networking & telecom. For him, the end result should be extendable, scalable, secure, and easy-to-use systems!

He believes in community ecosystems and tries to share his learning with larger audiences. He writes regularly at www.chandanpande.com, and this book is a step forward to connect with the community.

When he is not working or writing, he likes to go on road trips with his family to explore new avenues—be it spiritual, historical, or just a leisure tour! India being his home country, he is never short of surprises and variety!
Acknowledgments

I would like to thank the team at Packt Publishing. Richard, thanks a lot for finding my blog on Spring Integration and presenting me with an opportunity to share my knowledge in a more formal way and with a wider audience, and thanks for answering all of my questions with such patience—a few of which often took pages! I would like to thank Sharvari and Ritika, who are the content development editors for this book. Sharvari’s support and help with the technical review and incorporation of subsequent feedback was unprecedented. Aman Preet, the technical editor of the book, did a brilliant job with the final edit.

The greatest support is not what we see but the silent love and prayers of all our well-wishers. My parents’ blessings have always been with me. They are not technical and they do not understand what I do—but their unflinching confidence in me that whatever I am doing must be correct not only pumps up my confidence, but also puts a lot of responsibility on me. I hope I will not disappoint them ever.

Without a supportive family, it’s extremely difficult to achieve anything significant—and I consider myself lucky on that front. I am indebted to the charming smiles of my 2-year-old son, Aadish, and my 4-year-old daughter, Aashirya—their innocent smiles always remind me that happiness is beyond material success or anything else. As for my better half, Priyanka, I honestly believe that her work is more creditable than mine. Bringing up two small kids without any help from me, while at the same time supporting me throughout this assignment—a simple “thanks” will be too small a word for her, so I will leave it up to her to understand my gratitude!
Spring Integration Essentials

Software has been an integral part of enterprises and a key contributor to their growth. Be it analytics, user experience, social marketing, decision support systems, or any other functional domain, software has been used to aid smooth and efficient functioning. Enterprises start small and grow over a period of time, and so does their software dependency. Enterprise applications are developed over a period of time. The following aspects pose certain challenges while dealing with enterprise software:

- They are distributed across a set of domains, for example, payroll, inventory, reporting, and social integration.

- Each of these modules might have been developed independent of each other and may be on different platforms, for example, employee self-portal in J2EE stack, legacy records management on mainframes, CRM system using Salesforce, with some real-time application in their proprietary implementation.

- These modules need to interact with each other and with external systems as well. They may have to consume data from external sources through SOAP services or shared files, or they themselves have to share data though one of many data-sharing techniques.

- As software grows old, we need to introduce new platforms and replace existing modules to alleviate the growing maintenance cost. A rip and replace strategy would not work; rather, this should be done in a homogenous way without disturbing the sanity of existing modules during the transitions.

Integration of these modules either inside organizations or with external partners is inherently complex, requiring integration of heterogeneous endpoints. This is the kind of scenario that Enterprise Application Integration tries to address. Enterprise Integration Patterns (EIP) is a collection of standard enterprise challenges and how can they be handled. Spring Integration is one of the implementations of the EIP that provides many off-the-shelf components recommended by EIP.
How enterprise integration challenges can be solved

Many approaches have been tried to make the integration simple without compromising on vital aspects of enterprise, such as security, transaction, availability, reliability, and so on. A few of the prominent methodologies used over time are Java Connector Architecture (JCA), RMI, RPC, and CORBA for platform-agnostic integration, message brokers with system adapter, and many more. Under the hood, they try to solve integration issues through one of the following techniques:

- **Shared File**: This is the simplest approach. Systems can write data in a predefined format to a file, which can be read by other endpoints. An adapter might be needed to convert a format between two different endpoints. Let's consider an example, a daily report used to be generated in a CSV file. Over time, the organization established a web presence and reports need to be pushed online now. How can this be achieved? The simplest way is to dump it in files that will be read by an adapter and fed into the CMS system. A filesystem is simple but not the best solution; it is not transactional. What if a particular file is corrupt, or what if at poll interval files are not available due to network failure? This necessitates the introduction of a complex system that has a retry mechanism, filter capabilities, and many more nonfunctional aspects such as secure access, archival, and so on.

- **Shared database**: This addresses a few of the challenges that are addressed by the filesystem, such as transactional behavior, role-based access, performance tuning, distributed support, and so on. The common mechanism is a set of join tables—one application writes data in a schema that is understood by others. On the flip side, this introduces tight coupling; if there is a change in schema, both the systems need to be updated accordingly. Evolving applications will become a challenge, as they will have to factor in the external system limitations. The integration effort might itself start with lot of hiccups, for example, compatibility issues between the SQL provided by database vendors of the application, data format, and types in their table. For example, if one system stores only the date while the other stores the date with time stamp, depending on need, at least one will have to change format.
- **Remote procedure calls**: This mechanism introduces a paradigm where each system can offer services by publishing the contract. These paradigms can be a method, parameters, result, and error. For example, an EJB service or a SOAP service can be exposed for providing raw data for a reporting module that renders it in multiple formats. The most limiting aspect is synchronous behavior, where systems have to wait for the result. There are other challenges such as serialization of data, network latency, performance issues of a module, which can bring down the whole application, and so on. From a security aspect, exposing the method name and parameter invites hackers to exercise their creativity.

- **Messaging**: This introduces the asynchronous model in which two heterogeneous modules can interact through data over a predefined connection. The greatest advantage is decoupling—none of the systems are dependent on the availability of the other and they can participate or withdraw from integration without impacting other components. JMS is an example of message-based integration. Spring Integration is based on this paradigm where multiple endpoints connect on to a channel, produce or consume messages, and perform further processing based on information in a message. We will deal with channel, endpoints, message payload, and other concepts in the upcoming chapters.

Even if we use one of the preceding techniques, enterprise systems are way outward from each other and all of them might not be working all the time. This necessitated the use of middleware that can orchestrate reliable communication between these disparate endpoints, typically called an Enterprise Service Bus (ESB). In layman's terms, ESB can be defined as the middle man who enables communication to and fro between heterogeneous interfaces.

**Who are the players?**

As we have been discussing, the problem of enterprise integration is complex and many vendors have tried to address it in their own propitiatory ESB framework—earlier it used to be dominated by commercial vendors such as Tibco, Vitria, IBM MQSeries, Oracle SOA Suite, Microsoft BizTalk, and so on. Over time, the need for open source frameworks became evident as smaller organizations grew. Their integration needs were limited and were incapable of investing upfront with any of these biggies.
Some of the prominent open source integration frameworks, apart from Spring Integration, are Camel, Service Mix, Mule ESB, Open ESB, and so on. A comprehensive comparison of these frameworks is beyond the scope of this book but a small summary of two other major open source frameworks, has been provided here for the sake of emphasizing Spring Integration simplicity:

- **Mule ESB**: It is a standard server, solutions are developed and deployed inside them. Mule is one of the most prominent and stable solutions on the market. The point to be observed here is that, it's a container that holds the application.

- **Service Mix (SM)**: Apache Service Mix is built over JAVA legacy JBI (Java Business Integration). Service Mix tries to solve almost all aspects of enterprise integration by unifying the features and functionality of ActiveMQ, Camel, CXF, ODE, and Karaf. It provides a complete, enterprise-ready ESB, exclusively powered by OSGi. Since it tries to address a lot of modules, it is pretty bulky compared to Spring Integration.

### Why Spring Integration?

Spring Integration is an open source effort to address integration challenges; it is based on the Spring Framework, which is the most widely used Java-based framework in organizations. It introduces the simple POJO-based programming model to support standard integration patterns.

It's lightweight; all it needs is couple of jars for which Maven targets are readily available. A quick comparison shows that the Service Mix download is around 55 MB while Spring Integration is just 14 MB in size.

- Spring Integration is just a set of standard Java libraries; the solution gets deployed in the application instead of that application getting deployed in some containers, as in the case of SM and Mule.

For enterprises that are already using Java and Spring, it eases the integration effort as it follows the same idioms and patterns of the Spring Framework.

### What This Book Covers

*Chapter 1, Getting Started*, explains how to set up the Eclipse IDE, a "Hello World" program, and a brief introduction of how Spring ROO can ease the configuration aspects even further. This will help overcome configuration nightmares and warm up developers to a hands-on experience.
Chapter 2, Message Ingestion, introduces channels through which messages can be read and processed. It describes the point-to-point and pub-sub models, which one is best suited for a given scenario, how errors can be handled in a decoupled manner on a channel, and finally how in-memory channels can be backed up with persistence for failover and recovery solutions.

Chapter 3, Message Processing, explains how to define components that can apply business logic on messages, introduces decoupled logging that can used for auditing, and discusses adding transactional behavior.

Chapter 4, Message Transformers, deals with processing message formats, its conversion to a homogenuous format, and how annotations can help keep the configurations clean. Messages can be introduced in heterogeneous formats such as XML, JSON, and so on that need to be converted to a format understood by the system.

Chapter 5, Message Flow, will introduce flow aspects to messages such as filtering messages that do not comply to validation rules, routing them to an error branch, splitting messages, and redirecting them to components appropriate for their processing—waiting for incomplete payloads, aggregating partial messages, and finally the chaining of business processing handlers.

Chapter 6, Integration with External Systems, will give a hands-on overview of integration points. Integration with external systems is the most interesting and powerful aspect of Spring Integration—interaction with external systems is a matter of a few lines of configuration. Spring Integration has introduced adapters, gateways, and other components that make it a breeze to interact with filesystems, SQL, NoSQL persistence store, HTTP services, and other widely used external entities such as different servers, social media, and so on.

Chapter 7, Integration with Spring Batch, will introduce how to use Spring Integration and batch module for scheduling, triggering, and monitoring batch jobs.

Chapter 8, Testing Support, will explain how to leverage the readily available mocks for different components, what to test, and how much to test.

Chapter 9, Monitoring, Management, and Scaling Up, will cover using Spring Integration configuration to leverage JMX to get performance statistics of different configured components in the system. We will also peek into ways to scale up Spring Integration components.

Chapter 10, An End-to-End Example, has an end-to-end hands-on example that will help you to recollect concepts introduced in different chapters and reassert their understanding. Code will be pushed to a social repository as GitHub, but this chapter will give users enough instructions to use it and run it.
Getting Started

In this chapter, we will set up our development environment and discuss how we can leverage SpringSource Tool Suite (STS) to its maximum. Although any popular Java development IDE such as Eclipse, IntelliJ, NetBeans, and others can be used for developing Spring Integration solutions, pivotal, the company spearheading Spring Integration, recommends that you use STS which is an Eclipse-based IDE.

Setting up STS

STS comes with many off-the-shelf plugins, visual editors, and other features, which ease the development of Spring-powered enterprise applications. The look and feel of the IDE is very similar to Eclipse. Install STS by following these steps:

1. JDK 1.6 and above is a prerequisite, download and install it from http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html.
2. Set JAVA_HOME properties as explained in the documentation at https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/index.html.
4. The downloaded file is in ZIP format. Extract it to the preferred folder and it’s all set.
5. Go to <installation-directory>\sts-bundle\sts-3.6.1.RELEASE. The STS.exe file is the executable for launching the IDE.
6. This step is optional but can help in efficient functioning of the OS editor—change the memory allocation parameter. Locate STS.ini (in the same folder as STS.exe) and change the value of Xmx. For 2 GB, I’ve put it as Xmx2048m.
Creating your first project

The following steps will help you in creating your first project:

1. Create a Spring Integration project by navigating to File | Spring Project, as shown in the following screenshot:

2. Under the templates section, select Spring Integration Project - Simple. Provide a project name, for example, sisimple, as shown in the following screenshot:
3. Fill in the information required to create a Maven-based project, as shown in this screenshot:

![Maven Project Creation Screenshot]

4. Click on **Finish**; this will create a project with the name that was provided by us (**sisimple**), as shown in this screenshot:

![Maven Project Structure Screenshot]
This project is as simple as it can be. Let's take a quick look at the generated Java classes in the following points:

- **Main.java**: This file is located at the path: `/sisimple/src/main/java/com/chandan/example/si/`. It has the main method and will be used to run this sample. Right-click on this file from the package explorer and click on **Run As | Java Application** — this will start the program. This class has the code to bootstrap Spring Integration configuration files and load components defined in it. Additionally, it converts user input to upper case.

- **StringConversionService.java**: This file is located at the path: `/sisimple/src/main/java/com/chandan/example/si/service/`. This is the service interface that is used to convert user input to upper case.

- **spring-integration-context.xml**: This file is located at the path: `/sisimple/src/main/resources/META-INF/spring/integration/`. It is the Spring Integration configuration file. It contains the XML-based declaration of Spring Integration components.

- **log4j.xml**: This file is located at the path: `/sisimple/src/main/resources/`. It is the Log4j configuration file. It can be edited to control the log level, appenders, and other logging-related aspects.

- **StringConversionServiceTest.java**: This file is located at the path: `/sisimple/src/test/java/com/chandan/example/si/`. This is the test file for StringConversionService. This will be used to run tests against the service classes.

- **pom.xml**: This is the file used for maven dependency management, located in `/sisimple/`. It has entries for all the dependencies used by the project.

It will be a bit heavy and premature to explain each of the components in these classes and configuration files without having built up some theoretical concepts — we will discuss each of the elements in detail, as we move ahead in the chapters.
STS visual editor

STS provides visual ways to add different namespaces. Locate `spring-integration-context.xml` under `/sisimple/src/main/resources/META-INF/spring/integration/` and open it. This is the default Spring configuration file. Click on the Namespaces tab to manage different namespaces of Spring Integration. The following screenshot shows imported namespaces for this sample project:
In the same editor, clicking on the **Integration-graph** tab will open a visual editor, which can be used to add/modify or delete endpoints, channels, and other components of Spring Integration. The following screenshot contains the integration graph for our sample project:

Let’s have a quick look at the generated Maven POM—overall, there are three dependencies; only one for Spring Integration, and the other ones for Junit and log4j, as shown in the following screenshot:
Spring Integration Scala DSL

This is still in the very early stages and is an incubation project. Scala DSL should not be confused with other EIP implementations being offered in Scala—rather, it is built on top of Spring Integration and provides DSL-based configuration and flow management.

Check out the official Spring Integration Scala DSL blog at http://spring.io/blog/2012/03/05/introducing-spring-integration-scala-dsl/ and the GitHub page at https://github.com/spring-projects/spring-integration-dsl-groovy.

Summary

In this chapter, you learned how to set up your IDE and created a basic project. We also tried our hands at the visual editor of STS and covered a quick introduction of the upcoming Scala DSL for Spring Integration. We will leverage this knowledge to build a compelling Spring Integration application using STS throughout the rest of the chapters.

In the next chapter, we will cover how to ingest messages in the application and then how to process them.
Where to buy this book

You can buy Spring Integration 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.