Thoughts on Software design and development

Encounter and experiences during software development

Spring Integration -Getting started tutorial

OCT 29
Posted by **ChandanPandey**

i
7 Votes

A brief update, over past years I have continued working with spring integration and in quest to share my experience I have put challenges faced by me and it's programmatic solution in a book name spring integration essentials which is available at amazon (http://www.amazon.com/Spring-Integration-Essentials-Chandan-Pandey/dp/1783989165) and packtpub (https://www.packtpub.com/application-development/spring-integration-essentials). , and code samples at Github (https://github.com/cpandey05/siessentials)-please free to connect with me for any clarification or issues.

It's NOT a theory book, you can directly dive into using code samples (they are available on Github, so you can try experimenting with them right away)

Let's start with our Introduction 🙂
——————————————————————————————————————
I worked on one of the projects using spring integration. thought of sharing a quick hands on tutorial. I will cover following along with code snippet.

1. Introduction
   - What problem does Spring Integration solves –Enterprise Application Integration
   - A brief overview of Spring Integration
   - What are alternatives
   - Why Spring Integration over other
2. Getting Started

- Main components (Message, Message Channel, Message End Point)
- Message Endpoints (Transformer, Filter, Activator, Adapter)

3. Scaling UP
    - How to implement multiple executors
4. Infrastructural Support
    - JMX
5. IDE -STS
6. Basic hands On
    - A Example with REST based API and transformer

- **Introduction**
  Enterprise application have heterogeneous interfaces and applications, for example accounting, payroll, reporting, maintenance etc. which have been developed over a period of time with using different platforms (e.g. financial in COBOL, payroll in J2ee, maintenance in c/c++ etc.)

  Intercommunication between these modules have always been a challenge. How a c/c++ application talk with Java app, how CORBA and SOAP can be made seamless, how a File system and DB interaction be made possible -these all have been tried in multiple different ways. Enterprise Application Integration is an effort to address these kind of scenario.

  Major EAI approaches are

    - File Transfer: System interacts via files. For example an external provider can put files on a FTP
      server which contains input for other application.
    -
    - Shared Database: Two systems interact via same database. For example in a SaaS environment, catalogue of product can be stored at same DB location which different vendors can use.
    - Remote Procedure Call: We see this everyday –Avatars are SOAP, REST most talked and "Hot"
    - Messaging: Two words- JMS architecture

  enterprise pattern (http://www.eaipatterns.com/) defines major patterns for enterprise integration.
  Spring Integration is one of the implementation of the enterprise patterns.
  When we talk of heterogeneous communication, additional challenges comes in terms of synchronous and asynchronous communication mode. One of endpoints may not be available at time of interaction, or may be less responsive -this necessitates a need for an orchestrator which can retry, communication back, audit and do other stuff in role of a mediator. Here enters Enterprise Service Bus (ESB). ESB acts as mediator between these heterogeneous interfaces.

- **A brief introduction of Spring Integration:**
  Quoted from official Spring Integration (http://www.springsource.org/spring-integration)
  "Spring Integration provides an extension of the Spring programming model to support the well-known Enterprise Integration Patterns. It enables lightweight messaging within Spring-based applications and supports integration with external systems via declarative adapters. Those adapters provide a higher-level of abstraction over Spring's support for remoting, messaging, and scheduling. Spring Integration's primary goal is to provide a simple model for building enterprise integration solutions while maintaining the separation of concerns that is essential for producing maintainable, testable code."

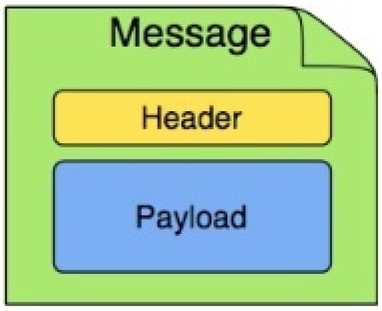
- **Alternatives:**
  As discussed earlier, there have been numerous attempts to address the EAI. It's difficult to list all – but two examples are.
  Mule ESB : It is a container, application need to be dropped in to leverage its support.
  Service Mix: It is another ESB, having a heavy footprint and trying to address lot of issues with lot of modules (OSGI Bundles)!
  Description of any of the above is beyond scope.

- **Why Spring Integration:**
  - Does not need a container or separate process space, can be invoked in existing program
  - Very light footprint, it is just a JAR which can be dropped with WAR or standalone systems
  - spring framework module, goes easily with existing spring and java project
- **Getting started**
  An ESB must support following components

  - Routing: It should be able to route messages across different endpoints based on configured rules or conditional logic.
  - Messaging: Convert message to and from different incompatible format to a normalized format.
  - Mediation: Making communication possible between heterogeneous interfaces.
  - Complex Event Processing (CEP): Capability to listen for events and trigger it as well.
  - Invocation: It should be able to orchestrate service interaction and invocation.

  It should also support logging, auditing, authentication (security), and management, etc, above and beyond the services listed above.

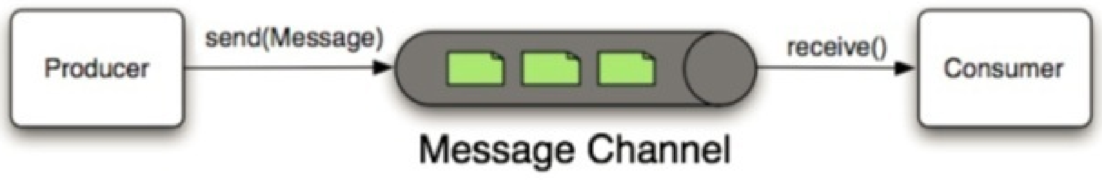
- **What are the major components supported by Spring:**
  (Definitions and images are standard and taken from Spring Integration Reference. The purpose is to provide quick reference)
  Message
  In Spring Integration, a Message is a generic wrapper for any Java object combined with metadata used
  by the framework while handling that object. It consists of a payload and headers. The payload can be of any type and the headers hold commonly required information such as id, timestamp, correlation id, and return address.



(https://chandanpandey.files.wordpress.com/2012/10/simessageformat1.jpg)

Message Channel
A Message Channel represents the "pipe" of a pipes-and-filters architecture. Producers send Messages to a channel, and consumers receive Messages from a channel. The Message Channel

therefore decouples the messaging components, and also provides a convenient point for interception and monitoring of Messages.



(https://chandanpandey.files.wordpress.com/2012/10/si_messagechannel1.jpg)
A Message Channel may follow either Point-to-Point or Publish/Subscribe semantics. With a Point-to-Point channel, at most one consumer can receive each Message sent to the channel. Publish/Subscribe channels, on the other hand, will attempt to broadcast each Message to all of its subscribers.

Should channels buffer the messages?
Pollable Channels are capable of buffering Messages within a queue. The advantage of buffering is that it allows for throttling the inbound Messages and thereby prevents overloading a consumer. On the other hand, a consumer connected to a Subscribable Channel is simply Message-driven.

- Message Endpoints
  A Message Endpoint represents the "filter" of a pipes-and-filters architecture. As mentioned above, the endpoint's primary role is to connect application code to the messaging framework and to do so in a noninvasive manner. In other words, the application code should ideally have no awareness of the Message objects or the Message Channels. This is similar to the role of a Controller in the MVC paradigm. Just as a Controller handles HTTP requests, the Message Endpoint handles Messages. Just as Controllers are mapped to URL patterns, Message Endpoints are mapped to Message Channels. The goal is the same in both cases: isolate application code from the infrastructure.

  Typical endpoints:

  - Transformer
    A Message Transformer is responsible for converting a Message's content or structure and returning the modified Message. Probably the most common type of transformer is one that converts the payload of the Message from one format to another (e.g. from XML Document to java.lang.String).

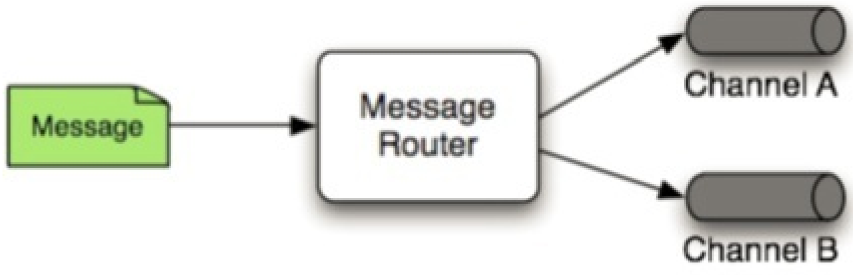
    Typical use case : Convert XML and File based payloads to Domain objects.

  - Filter
    A Message Filter determines whether a Message should be passed to an output channel at all. This simply requires a boolean test method that may check for a particular payload content type, a property value, the presence of a header, etc.

    Typical use case: Drop all message which does not meet the minimum criteria

  - Router
    A Message Router is responsible for deciding what channel or channels should receive the Message next (if any).

Typical use case: If a XML is parsed successfully, it is passed on to next channel for further processing else is routed on a retry channel.



(https://chandanpandey.files.wordpress.com/2012/10/messagerouter3.jpg)

- Splitter
  A Splitter is another type of Message Endpoint whose responsibility is to accept a Message from its input channel, split that Message into multiple Messages, and then send each of those to its output channel.

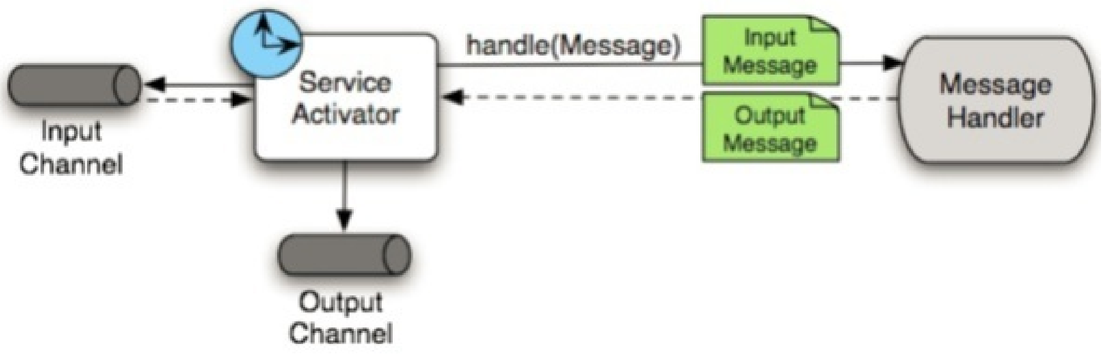
  Typical use case: After fetching list of files from a remote FTP server, list of files is put on a splitter, which then puts it on a gateway –one by one, so that each file can be fetched.

- Aggregator
  Basically a mirror-image of the Splitter, the Aggregator is a type of Message Endpoint that receives multiple Messages and combines them into a single Message. Spring Integration provides a completion strategy as well as configurable settings for timeout, whether to send partial results upon timeout, and the discard channel.

  Typical use case: PDF and related images are available from FTP server. Aggregator is used to wait for both (PDF and associated JPEG) of them to be available –correlation strategy decides if both are available or not. Once available it is passed on for further processing.

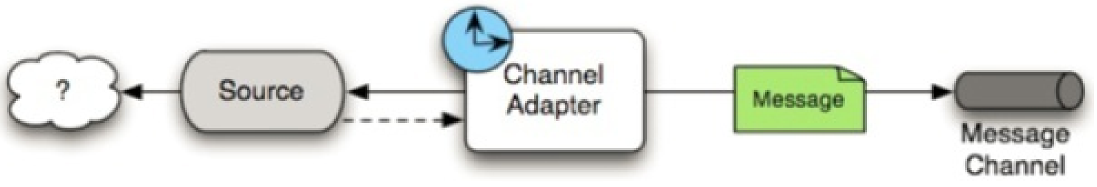
- Service Activator
  A Service Activator is a generic endpoint for connecting a service instance to the messaging system. The input Message Channel must be configured, and if the service method to be invoked is capable of returning a value, an output Message Channel may also be provided. The output channel is optional, since each message may also provide its own 'Return Address' header. This same rule applies for all consumer endpoints channel specified in the Message's "return address" if available.

  Typical use case: All business processing done in service activator, for example persistence.

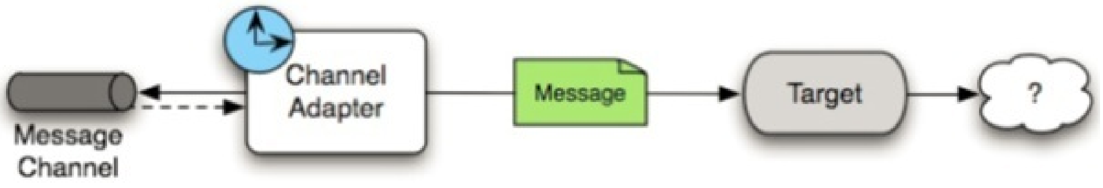


(https://chandanpandey.files.wordpress.com/2012/10/siserviceactivator1.jpg)

- Channel Adapter
  A Channel Adapter is an endpoint that connects a Message Channel to some other system or transport. Channel Adapters may be either inbound or outbound.

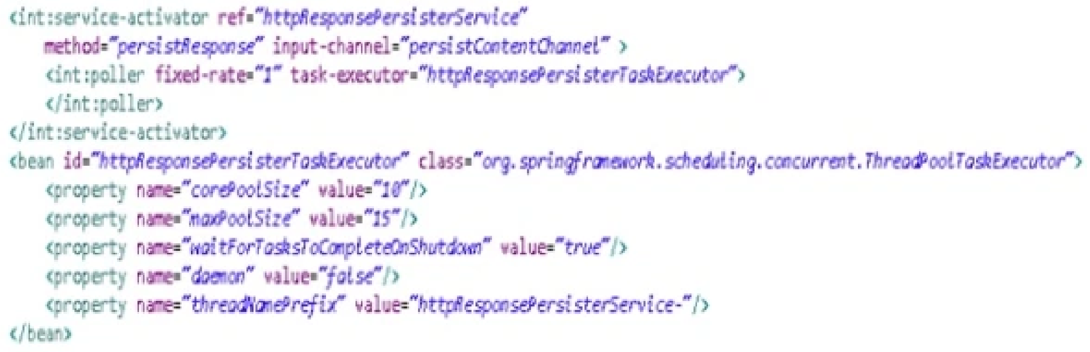
  

  (https://chandanpandey.files.wordpress.com/2012/10/siinboundchanneladapter1.jpg) An inbound "Channel Adapter" endpoint connects a source system to a MessageChannel

  (https://chandanpandey.files.wordpress.com/2012/10/sioutboundchanneladapter1.jpg)
  An outbound "Channel Adapter" endpoint connects a MessageChannel to a target system.

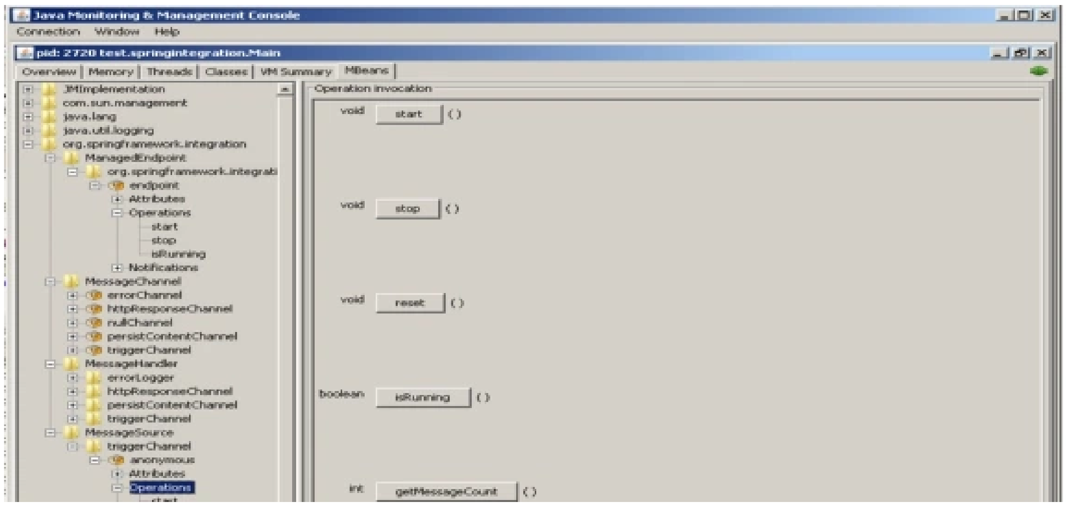
- **Threading support:**
  Multiple instances of service activator can be configured to work on same channel. This will help in avoiding build up of channel. For optimal performance of threading, polling interval and max messages per poll should be set appropriately. Lets say, requirement is to process ~7k files in an hour–thread pool executor have been implemented to process items parallel y. A sample code:

```
<int:service-activator ref="httpResponsePersisterService"
    method="persistResponse" input-channel="persistContentChannel" >
    <int:poller fixed-rate="1" task-executor="httpResponsePersisterTaskExecutor">
    </int:poller>
</int:service-activator>
<bean id="httpResponsePersisterTaskExecutor" class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
    <property name="corePoolSize" value="10"/>
    <property name="maxPoolSize" value="15"/>
    <property name="waitForTasksToCompleteOnShutdown" value="true"/>
    <property name="daemon" value="false"/>
    <property name="threadNamePrefix" value="httpResponsePersisterService-"/>
</bean>
```

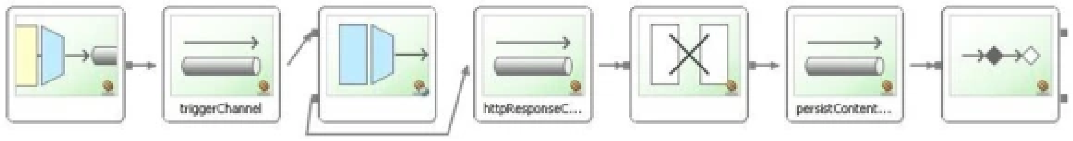
  (https://chandanpandey.files.wordpress.com/2012/10/scalingup.jpg)

- **Infrastructural Support**
  - **JMX**
  - Sing line of configuration exposes all of the channels on standard JMX console.
  - Sample code snippet:
    - ‹context:mbean-server/›
    - ‹int-jmx:mbean-export id="integrationMbeanExporter"/›



(https://chandanpandey.files.wordpress.com/2012/10/sijmx1.jpg)

- **STS Visual Editor –**
  It can depict the Integration flow diagrammatically.



(https://chandanpandey.files.wordpress.com/2012/10/siflow2.jpg)

  - An in bound channel adapter post dummy message on trigger channel
  - HTTP outbound gateway listens on trigger channel and initiates a HTTP request as soon as channel gets some data
  - HTTP outbound gateway writes its response to HTTP response channel.
  - Transformer is connected to HTTP response channel, which transforms the message and passes on to a persister channel
  - Service Activator takes input from persister channel and performs business logic.
- **Basic hands On -A small code example**
  It is a code example as per above flow. Download (https://github.com/cpandey05/codeexamples/tree/master/spring-integration-sample-app)the code hosted at Github. It is a maven project -download it and import it as a maven project. Details of class files:

  - Main.java : main class, this boots up the application.
  - InboundAdapter.java : Post dummy message on the trigger channel
  - SimpleTransformer.java :Transformer, which transforms the message –it just adds some custom string. It can be more meaning full as transforming to domain object.
  - HttpResponsePersisterService.java : Service activator –here business logic as persistence can go. For e.g. have just logged it in a log file.
  - spring-sample-app\src\main\resources\META-INF\application-context-root.xml : All the configurations for adapter, transformer and service activator here. Quite self explanatory
  - spring-sample-app\src\main\resources\log4j.properties : Log file settings.

——

A brief update, over past years I have continued working with spring integration and in quest to share my experience I have put challenges faced by me and it’s programmatic solution in a book name spring integration essentials which is available at amazon (http://www.amazon.com/Spring-Integration-Essentials-Chandan-Pandey/dp/1783989165) and packtpub (https://www.packtpub.com/application-development/spring-integration-essentials). , and code samples at Github (https://github.com/cpandey05/siessentials)-please free to connect with me for any clarification or issues.

It's NOT a theory book, you can directly dive into using code samples (they are available on Github, so you can try experimenting with them right away)

About ChandanPandey

Try to come up with a good design as by product of good coding practices

View all posts by ChandanPandey »

Posted on October 29, 2012, in Coding practice, Spring, Spring Integration and tagged code sample, Development, spring, Spring Integration tutorial, tutorial. Bookmark the permalink. 6 Comments.

- ### Leave a comment

- ### Trackbacks 1

- ### Comments 5

Ingela | September 3, 2013 at 1:49 pm

1

0

i
Rate This

Could not download code on Github. 404 This is not the webpage you are looking for.

ChandanPandey | September 3, 2013 at 2:01 pm

1

0

i
Rate This

Thanks for pointing. Please try now -I have corrected the link.

Steve | October 26, 2013 at 9:04 pm

0

0

i
Rate This

Very good article. Thank you.
I'm new to Spring integration – I have a Cloud server that I would like to transfer files to and from using Spring Integration. What setup do I need on the cloud server?

Thanks!!!

ChandanPandey | October 28, 2013 at 1:11 pm

1

0

i
Rate This

You will need to open port for FTP from the security group(Amazon), and then can use FTP adapters. Other way, you can write service activator which can do ssh to the cloud -ssh port is by default open on most of the cloud servers

Abhishek | March 28, 2014 at 3:13 pm

1

0

i
Rate This

can you provide an example of reading file from database and write into file using spring integration.

1. Pingback: **What is spring integration and how can it help solve enterprise integration challenges | Thoughts on Software design and development**