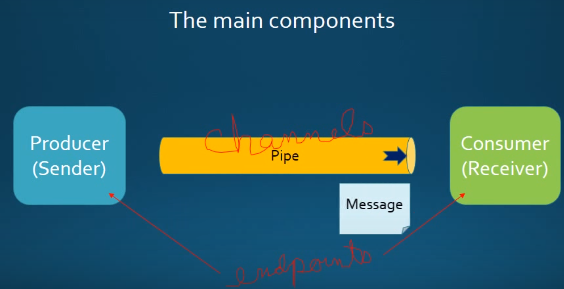
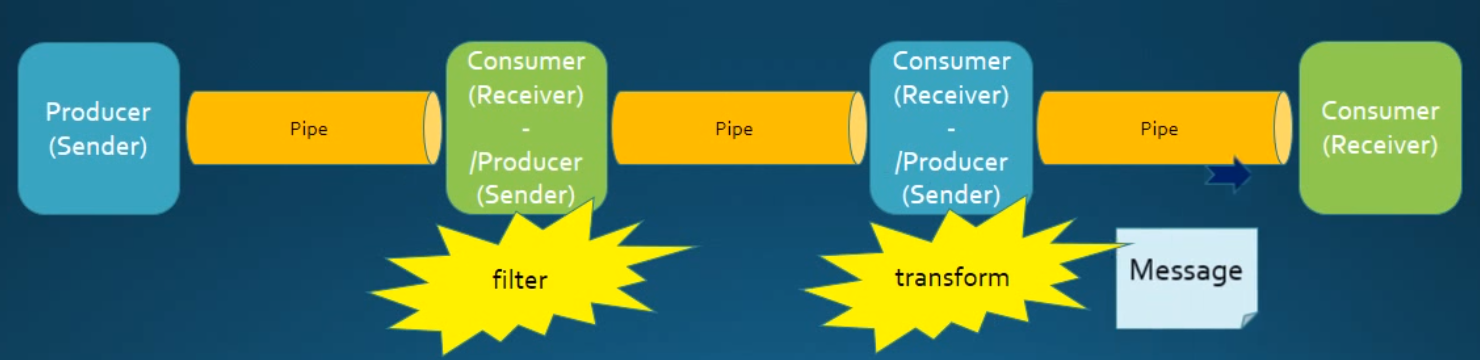
Spring Integration

# Introduction

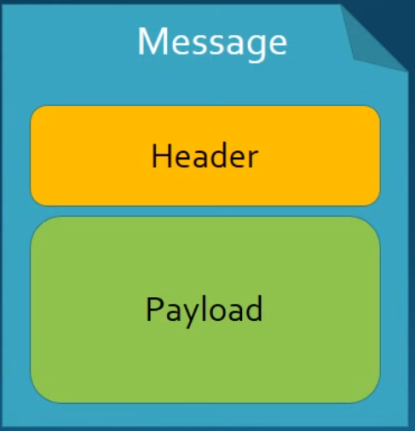
The main components: **Messages, Channels** and **Endpoints**.



Chains and pointing channels together



## Messages

**Header**: contains system information, e.g. creation timestamp

**Payload**: contains the data

## Message Endpoints

* **Adapters**: Connect your channel to some other system
* **Filter**: remove some messages from channels based on header, content, etc)
* **Transformer**: convert a message content or structure
* **Enricher**: add conent to the message header or payload
* **Service** activator: invoke service operations based on the arrival of a message.
* **Gateway**: connect your channels without spring integration coupling.

## Message Channels

**public** **interface** MessageChannel {

*//Send a Message to this channel. If successfully, returns true.*

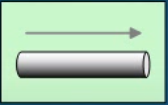
**boolean** send(Message<?> message);

*//Send a Message, blocking until either accepted or timeout elapses.*

**boolean** send(Message<?> message, **long** timeout);

}

The icon in Enterprise Integration Patterns(EIP) diagrams:



Two general classifications of message channels.

### Pollable Channel(buffering)

**public** **interface** PollableChannel **extends** MessageChannel {

*// Receive a message from this channel, blocking indefinitely if*

*// necessary.*

Message<?> receive();

*// Receive a message from this channel, blocking until either a message*

*// is available or the specified timeout period elapses and return null*

Message<?> receive(**long** timeout);

}

Pollable Channel may buffer its messages.

* Requires a queue to hold the messages
* The queue has a designated capacity

Waits for the consumer to get the messages, consumers activately poll to receive messages.

It is typically a point-to-point channel, only one receiver of a message in the channel. Usually used for Document or information.

### Subscribable Channel(non-buffering)

*// A MessageChannel that maintains a registry of subscribers and invokes*

*// them to handle messages sent through this channel.*

**public** **interface** SubscribableChannel **extends** MessageChannel {

*//Register a message handler.*

**boolean** subscribe(MessageHandler handler);

*//Un-register a message handler.*

**boolean** unsubscribe(MessageHandler handler);

}

Subscribable channel allows multiple subscribers (or consumers) to register for its messages.

* Messages are delivered to all registered subscribers on message arrival
* It has to manage a list or registry of subscribers

It doesn’t buffer its messages. Usually used for “event” messages.

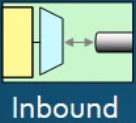
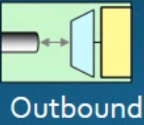
# Adapters

Adapters: are the endpoints in spring integration that connects channel to an actual system. It provides the bridge between integration framework and the external systems and services(bootk: pro spring integration.)

It provides separation of concerns, that helps to separate the messaging API from what is to transport and protocol used entire spring integration system. You don’t want your code have to know a lot about JMS or JDBC, the spring intgration adapters help provide for those capabilities.

Adapters are classified as either inbound or outbound adapters

* **Inbound Adapter**: bring messages into the spring integration channels
* **Outbound Adapter**: Get the messages out at the Spring Integration Channels and into outside the applications databases etc.

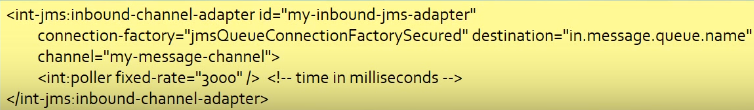
Built-in adapters:

* Stream adapters
* File Adapters
* JDBC & JPA Adapters
* FTP and Secure FTP (SFTP) Adapters
* Feed(RSS, Atom, etc.) Adapters
* Mail Adapters
* MongoDB Adapters
* UDP Adapters
* Tweeter Adapter

## A JMS Inbound Adapter

The following configuration takes messages from a message Queue (via JMS under the covers) and gets it into a spring integration channel.

This inbound adapter requires a poll of the messages into the channel. It means how often should this adapter poll messages into one of the channels.



## A JMS Outbound Adapter

The following configuration takes messages from a message channel and delivers it to a message Queue (via JMS unter the covers).



Diagram for the two examples together:

