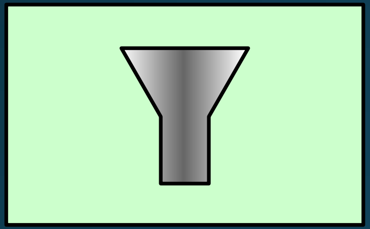


Filters

Express Spring Integration

SI Adapters Review

- An endpoint that connects a channel to an external system.
  - It "provide the bridge between integration framework and the external systems and services"[1]
  - Providing separation of the messaging concerns from the transports and protocols used.
- Adapters are inbound or outbound.
  - Those that bring messages into the SI channels.
  - Those that get messages from SI channels to the outside applications, databases, etc.
- SI provides many adapters out of the box.
  - File
  - JMS
  - Stream
  - Mail
  - FTP
  - etc.

Filters

- Spring Integration **filters** are endpoints that sit between channels and allow or reject messages from one message channel to the next.
    - Filters allow some messages to pass from one channel to another channel.
    - Messages not selected are discarded.
    - Selection occurs on the basis of message payload or message metadata (header information).
    - The logic of a filter is simple. It must either “accept” or “reject” a message coming from one channel to the next.
- As with adapters, SI provides many filters out of the box.
- You can create your own custom filter with its own custom message selection criteria.
- Filters are represented by this icon in EIP diagrams.

SI Built-In Filters

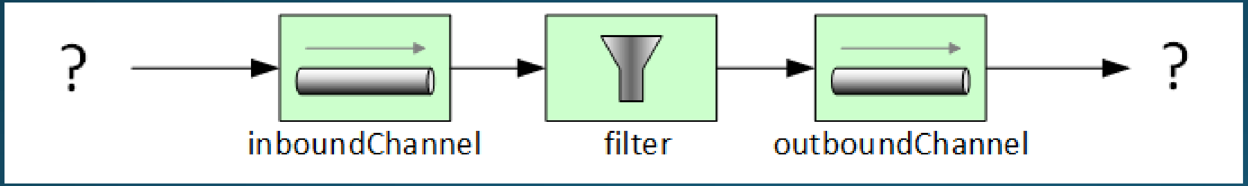
- Spring Integration provides several ready-to-use filters with the framework.
  - You merely have to configure them to use them.
- Built-in filters include:
  - Expression Filter – a filter that uses an evaluated SpEL expression against the message to select messages
  - Xpath Filter – Use Xpath expressions against the XML payload to select messages
  - XML Validating Filter – select XML payload messages that validate against a given schema.

Example Filter – a String Payload Filter

- Here is a filter that accepts all (String) messages that do not start with the text "Hello".

```
<int:filter input-channel="inboundChannel"  output-channel="outboundChannel"
      expression="payload.startsWith('Hello')" />
```

- A filter sits between two channels – as represented in this EIP diagram.



- Messages that are rejected are simply removed from the system.
  - Optionally, a discard-channel can be specified with a filter to capture and route discarded

```
<int:filter input-channel="inboundChannel"  output-channel="outboundChannel"
      discard-channel="relook-channel"   expression="payload.startsWith('Hello')" />
```

Custom Filters

- To create a custom filter, you must implement the SI provided MessageSelector interface.
    - The MessageSelector’s method must have a method that returns a boolean indicating selection or rejection of each message it is passed.

```
public class MySelector implements MessageSelector {

        public boolean accept(Message<?> message) {
                if (message.getPayload() instanceof String
                                && ((String) message.getPayload()).startsWith("Hello"))
                        return true;
                return false;
        }
}
```

Filter/MessageSelector Configuration

- Once the MessageSelector is defined, configure a filter to use its logic to do the filtering.

```
<int:filter input-channel="inboundChannel" output-channel="outboundChannel"
        ref="selector" />
<bean id="selector" class="com.intertech.MySelector" />
```

You are ready to tackle Lab 3

- In Lab 3, you work with a couple of Spring Integration's built-in filters and also create a custom Filter (with MessageSelector)
- You also see a message transformer – something you'll learn more about in the next tutorial.

- Click here for associated labs and video

Associated Courses and Resources

Upcoming Training

- Spring Training
- To learn more: http://bit.ly/1hyrViM