FREE eBook

LEARNING spring-integration

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-integration

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: spring-integration

It is an unofficial and free spring-integration ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official spring-integration.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with spring-integration

Remarks

This section provides an overview of what spring-integration is, and why a developer might want to use it.

It should also mention any large subjects within spring-integration, and link out to the related topics. Since the Documentation for spring-integration is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
4.3.x	2016-11-07
4.2.x	2016-11-07
4.1.x	2016-07-25
4.0.x	2016-07-26
3.0.x	2015-10-27
2.2.x	2016-01-27
2.1.x	2013-06-10
2.0.x	2013-04-11
1.0.x	2010-04-16

Examples

Installation or Setup

The best way to get started using Spring-Integration in your project is with a dependency management system, like gradle.

```
dependencies {
    compile 'org.springframework.integration:spring-integration-core:4.3.5.RELEASE'
}
```

Below is a very simple example using the gateway, service-activator message endpoints.

```
//these annotations will enable Spring integration and scan for components
@Configuration
@EnableIntegration
@IntegrationComponentScan
public class Application {
    //a channel has two ends, this Messaging Gateway is acting as input from one side of
inChannel
    @MessagingGateway
    interface Greeting {
        @Gateway(requestChannel = "inChannel")
        String greet(String name);
    }

    @Component
    static class HelloMessageProvider {
        //a service activator act as a handler when message is received from inChannel, in
this example, it is acting as the handler on the output side of inChannel
        @ServiceActivator(inputChannel = "inChannel")
        public String sayHello(String name) {
            return "Hi, " + name;
        }
    }

    @Bean
    MessageChannel inChannel() {
        return new DirectChannel();
    }

    public static void main(String[] args) {
        ApplicationContext context = new
AnnotationConfigApplicationContext(Application.class);
        Greeting greeting = context.getBean(Greeting.class);
        //greeting.greet() send a message to the channel, which trigger service activitor to
process the incoming message
        System.out.println(greeting.greet("Spring Integration!"));
    }
}
```

It will display the string `Hi, Spring Integration!` in the console.

Of course, Spring Integration also provides xml-style configuration. For the above example, you can write such following xml configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:int="http://www.springframework.org/schema/integration"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/integration
        http://www.springframework.org/schema/integration/spring-integration.xsd">
    <int:gateway default-request-channel="inChannel"
                 service-
interface="spring.integration.stackoverflow.getstarted.Application$Greeting"/>
    <int:channel id="inChannel"/>
    <int:service-activator input-channel="inChannel" method="sayHello">
        <bean
```

```
class="spring.integration.stackoverflow.getstarted.Application$HelloMessageProvider"/>
    </int:service-activator>
</beans>
```

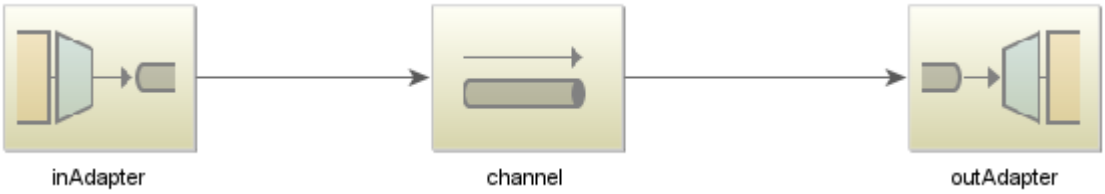
To run the application using the xml config file, you should change the code `new AnnotationConfigApplicationContext(Application.class)` in `Application` class to `new ClassPathXmlApplicationContext("classpath:getstarted.xml")`. And run this application again, you can see the same output.

Generic Inbound and Outbound Channel Adapter

Channel adapter is one of message endpoints in Spring Integration. It is used for unidirectional message flow. There are two types of channel adapter:

Inbound Adapter: input side of the channel. Listen or actively read message.

Outbound Adapter: output side of the channel. Send message to Java class or external system or protocol.



- Source code.

```
public class Application {
    static class MessageProducer {
        public String produce() {
            String[] array = {"first line!", "second line!", "third line!"};
            return array[new Random().nextInt(3)];
        }
    }

    static class MessageConsumer {
        public void consume(String message) {
            System.out.println(message);
        }
    }

    public static void main(String[] args) {
        new
ClassPathXmlApplicationContext("classpath:spring/integration/stackoverflow/ioadapter/ioadapter.xm

    }
}
```

- XML-style Configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
       xmlns:int="http://www.springframework.org/schema/integration"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/integration
http://www.springframework.org/schema/integration/spring-integration.xsd">
    <int:channel id="channel"/>
    <int:inbound-channel-adapter id="inAdapter" channel="channel" method="produce">
        <bean
class="spring.integration.stackoverflow.ioadapter.Application$MessageProducer"/>
        <int:poller fixed-rate="1000"/>
    </int:inbound-channel-adapter>
    <int:outbound-channel-adapter id="outAdapter" channel="channel" method="consume">
        <bean
class="spring.integration.stackoverflow.ioadapter.Application$MessageConsumer"/>
    </int:outbound-channel-adapter>
</beans>
```

- Message Flow
    - `inAdapter`: an inbound channel adapter. Invoke `Application$MessageProducer.produce` method every 1 second (`<int:poller fixed-rate="1000"/>`) and send the returned string as message to the channel `channel`.
    - `channel`: channel to transfer message.
    - `outAdapter`: an outbound channel adapter. Once message reached on channel `channel`, this adapter will receive the message and then send it to `Application$MessageConsumer.consume` method which print the message on the console.
    - So you can observe that these random choose string will displayed on the console every 1 second.

Simple Echo Excample with Spring-Integration-Stream

Java code:

```
public class StdioApplication {
    public static void main(String[] args) {
        new
ClassPathXmlApplicationContext("classpath:spring/integration/stackoverflow/stdio/stdio.xml");
    }
}
```

Xml config file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:int="http://www.springframework.org/schema/integration"
       xmlns:int-stream="http://www.springframework.org/schema/integration/stream"
```

```
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/integration/stream
       http://www.springframework.org/schema/integration/stream/spring-integration-stream.xsd
       http://www.springframework.org/schema/integration
       http://www.springframework.org/schema/integration/spring-integration.xsd">
    <int:channel id="channel"/>
    <int-stream:stdin-channel-adapter id="stdin" channel="channel">
        <int:poller fixed-rate="1000"/>
    </int-stream:stdin-channel-adapter>
    <int-stream:stdout-channel-adapter id="stdout" channel="channel"/>
</beans>
```

This is a echo example. When you run this Java application, you can input some string and then it will be displayed on the console.

Read Getting started with spring-integration online: https://riptutorial.com/spring-integration/topic/6985/getting-started-with-spring-integration

Chapter 2: Jdbc Integration

Examples

Jdbc Inbound Adapter - xml configuration

In the official reference document, it says:

> The main function of an inbound Channel Adapter is to execute a SQL SELECT query and turn the result set as a message. The message payload is the whole result set, expressed as a List, and the types of the items in the list depend on the row-mapping strategy that is used. The default strategy is a generic mapper that just returns a Map for each row in the query result.

- Source code

```
public class Application {
   static class Book {
      String title;
      double price;

      Book(String title, double price) {
          this.title = title;
          this.price = price;
      }

      double getPrice() {
          return price;
      }

      String getTitle() {
          return title;
      }

      @Override
      public String toString() {
          return String.format("{title: %s, price: %s}", title, price);
      }
    }

    static class Consumer {
      public void consume(List<Book> books) {
          books.stream().forEach(System.out::println);
      }
    }
```

```
    static class BookRowMapper implements RowMapper<Book> {

      @Override
      public Book mapRow(ResultSet rs, int rowNum) throws SQLException {
          String title = rs.getString("TITLE");
          double price = rs.getDouble("PRICE");
          return new Book(title, price);
      }
    }

    public static void main(String[] args) {
      new ClassPathXmlApplicationContext(
              "classpath:spring/integration/stackoverflow/jdbc/jdbc.xml");
    }
}
```

- xml configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:int="http://www.springframework.org/schema/integration"
       xmlns:int-jdbc="http://www.springframework.org/schema/integration/jdbc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/integration
http://www.springframework.org/schema/integration/spring-integration.xsd
       http://www.springframework.org/schema/integration/jdbc
http://www.springframework.org/schema/integration/jdbc/spring-integration-jdbc.xsd
       http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc.xsd">
    <jdbc:embedded-database id="dataSource" type="H2">
        <jdbc:script
location="classpath:spring/integration/stackoverflow/jdbc/schema.sql"/>
    </jdbc:embedded-database>
    <bean id="bookRowMapper"
          class="spring.integration.stackoverflow.jdbc.Application$BookRowMapper"/>

    <int:channel id="channel"/>

    <int-jdbc:inbound-channel-adapter id="jdbcInbound"
                                      channel="channel"
                                      data-source="dataSource"
                                      query="SELECT * FROM BOOKS"
                                      row-mapper="bookRowMapper">
        <int:poller fixed-rate="1000"/>
    </int-jdbc:inbound-channel-adapter>

    <int:outbound-channel-adapter id="outbound" channel="channel" method="consume">
        <bean class="spring.integration.stackoverflow.jdbc.Application$Consumer"/>
    </int:outbound-channel-adapter>
</beans>
```

- schema.sql

```
CREATE TABLE BOOKS (
  TITLE VARCHAR(20) NOT NULL,
```

```
  PRICE DOUBLE      NOT NULL
);

INSERT INTO BOOKS(TITLE, PRICE) VALUES('book1', 10);
INSERT INTO BOOKS(TITLE, PRICE) VALUES('book2', 20);
```

- Summary:
    - `jdbcInbound`: a Jdbc inbound channel adapter. It execute the SQL `SELECT * FROM BOOKS` and convert the result set to `List<Book>` via the bean `bookRowMapper`. Finally, it send this book list to the channel `channel`.
    - `channel`: transfer the message
    - `outbound`: a generic outbound adapter. see Generic Inbound and Outbound Channel Adapter

Jdbc Outbound Channel Adapter - xml configuration

In the Spring Integration Reference Docuement, it says:

> The outbound Channel Adapter is the inverse of the inbound: its role is to handle a message and use it to execute a SQL query. The message payload and headers are available by default as input parameters to the query...

- Java code

```
public class OutboundApplication {
    static class Book {
        String title;
        double price;

        Book(String title, double price) {
            this.title = title;
            this.price = price;
        }

        public double getPrice() {
            return price;
        }

        public String getTitle() {
            return title;
        }

    }

    static class Producer {
        public Book produce() {
```

```
            return IntStream.range(0, 3)
                    .mapToObj(i -> new Book("book" + i, i * 10))
                    .collect(Collectors.toList())
                    .get(new Random().nextInt(3));
        }
    }

    public static void main(String[] args) {
        new ClassPathXmlApplicationContext(
                "classpath:spring/integration/stackoverflow/jdbc/jdbc-outbound.xml");
    }
}
```

- xml config file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jdbc="http://www.springframework.org/schema/jdbc"
       xmlns:int="http://www.springframework.org/schema/integration"
       xmlns:int-jdbc="http://www.springframework.org/schema/integration/jdbc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/jdbc
       http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
       http://www.springframework.org/schema/integration
       http://www.springframework.org/schema/integration/spring-integration.xsd
       http://www.springframework.org/schema/integration/jdbc
       http://www.springframework.org/schema/integration/jdbc/spring-integration-
jdbc.xsd">
    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="url" value="jdbc:h2:tcp://localhost/~/booksystem"/>
        <property name="username" value="sa"/>
        <property name="password" value=""/>
        <property name="driverClassName" value="org.h2.Driver"/>
    </bean>
    <jdbc:initialize-database>
        <jdbc:script
location="classpath:spring/integration/stackoverflow/jdbc/schema.sql"/>
    </jdbc:initialize-database>
    <int:channel id="channel"/>
    <int:inbound-channel-adapter channel="channel" method="produce" >
        <bean
class="spring.integration.stackoverflow.jdbc.OutboundApplication$Producer"/>
        <int:poller fixed-rate="1000"/>
    </int:inbound-channel-adapter>
    <int-jdbc:outbound-channel-adapter id="jdbcOutbound"
                                       channel="channel"
                                       data-source="dataSource"
                                       sql-parameter-source-factory="sqlParameterSource"
                                       query="INSERT INTO BOOKS(TITLE, PRICE)
VALUES(:title, :price)"/>
    <bean id="sqlParameterSource"
class="org.springframework.integration.jdbc.ExpressionEvaluatingSqlParameterSourceFactory">

        <property name="parameterExpressions">
            <map>
                <entry key="title" value="payload.title"/>
```

```
                <entry key="price" value="payload.price"/>
            </map>
        </property>
    </bean>
</beans>
```

- schema.sql

```
DROP TABLE IF EXISTS BOOKS;
CREATE TABLE BOOKS (
  TITLE VARCHAR(20) NOT NULL,
  PRICE DOUBLE      NOT NULL
);
```

- You can observe the `BOOKS` table, and you can see the records are inserted. Or you can write a `int-jdbc:inbound-channel-adapter` to count the `BOOKS` table and you can find that the count number is growing continuously.

- Summary:

    - `inbound`: a generic inbound adapter used to get the `Book` object as message payload and send it to channel `channel`.
    - `channel`: used to transfer message.
    - `jdbcOutbound`: a jdbc outbound adapter, it receives the message with `Book` type and then prepare the query parameter `:title` and `:price` via `sqlParameterSource` bean using SpEL like `payload.title` and `payload.price` to get the title and price form the message payload.

Read Jdbc Integration online: https://riptutorial.com/spring-integration/topic/8140/jdbc-integration

Credits

S. No	Chapters	Contributors
1	Getting started with spring-integration	Community, Maxi Wu, walsh
2	Jdbc Integration	walsh