Kostenloses eBook

LERNEN spring-integration

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-integration

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit der Spring-Integration

Bemerkungen

In diesem Abschnitt erhalten Sie einen Überblick darüber, was Spring Integration ist und warum ein Entwickler sie verwenden möchte.

Es sollte auch alle großen Themen der Frühjahrsintegration erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation zur Frühjahrsintegration neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Versionen

Ausführung	Veröffentlichungsdatum
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 oder Setup

Um Spring-Integration in Ihrem Projekt einzusetzen, verwenden Sie am besten ein Abhängigkeitsverwaltungssystem wie Gradle.

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

Im Folgenden finden Sie ein sehr einfaches Beispiel für die Gateway- , Service-Aktivator-Nachrichtenendpunkte.

```
//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!"));
    }
}
```

Es wird die Zeichenfolge `Hi, Spring Integration!` angezeigt `Hi, Spring Integration!` in der Konsole

Natürlich bietet Spring Integration auch eine XML-Konfiguration. Für das obige Beispiel können Sie die folgende XML-Konfigurationsdatei schreiben.

```
<?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>
```

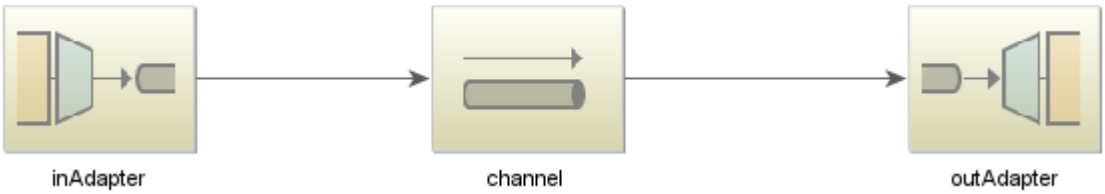
Um die Anwendung mit der XML-Konfigurationsdatei auszuführen, sollten Sie den Code `new AnnotationConfigApplicationContext(Application.class)` in `Application` Klasse in `new ClassPathXmlApplicationContext("classpath:getstarted.xml")` . Wenn Sie diese Anwendung erneut ausführen, können Sie dieselbe Ausgabe sehen.

Generischer Inbound- und Outbound-Channel-Adapter

Der Kanaladapter ist einer der Nachrichtenendpunkte in Spring Integration. Sie wird für den unidirektionalen Nachrichtenfluss verwendet. Es gibt zwei Arten von Kanaladaptern:

Inbound Adapter : Eingangsseite des Kanals. Nachricht abhören oder aktiv lesen.

Outbound Adapter : Ausgangsseite des Kanals. Nachricht an Java-Klasse oder externes System oder Protokoll senden.



- Quellcode.

```
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-Konfigurationsdatei:

```
<?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>
```

- Nachrichtenfluss
    - `inAdapter` : ein Inbound-Channel-Adapter. Rufen Sie die `Application$MessageProducer.produce` Methode alle 1 Sekunde auf (`<int:poller fixed-rate="1000"/>`) und senden Sie die zurückgegebene Zeichenfolge als Nachricht an den Channel- `channel` .
    - `channel` : Kanal zum Übertragen der Nachricht.
    - `outAdapter` : ein ausgehender `outAdapter` . Sobald die Nachricht auf dem Channel- `channel` , empfängt dieser Adapter die Nachricht und sendet sie an die `Application$MessageConsumer.consume` Methode, die die Nachricht auf der Konsole `Application$MessageConsumer.consume` .
    - Sie können also feststellen, dass diese zufällig ausgewählte Zeichenfolge alle 1 Sekunde auf der Konsole angezeigt wird.

Einfaches Echo-Beispiel mit Spring-Integration-Stream

Java-Code:

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

XML-Konfigurationsdatei

```
<?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>
```

Dies ist ein Echo-Beispiel. Wenn Sie diese Java-Anwendung ausführen, können Sie einen String eingeben, der dann in der Konsole angezeigt wird.

Erste Schritte mit der Spring-Integration online lesen: https://riptutorial.com/de/spring-integration/topic/6985/erste-schritte-mit-der-spring-integration

Kapitel 2: Jdbc-Integration

Examples

Jdbc Inbound Adapter - XML-Konfiguration

Im offiziellen Referenzdokument heißt es:

> Die Hauptfunktion eines Inbound Channel Adapters besteht darin, eine SQL SELECT-Abfrage auszuführen und die Ergebnismenge als Nachricht anzuzeigen. Die Nachrichtennutzlast ist die gesamte Ergebnismenge, ausgedrückt als Liste. Die Arten der Elemente in der Liste hängen von der verwendeten Zeilenzuordnungsstrategie ab. Die Standardstrategie ist ein generischer Mapper, der nur eine Map für jede Zeile im Abfrageergebnis zurückgibt.

- Quellcode

```
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-Konfigurationsdatei

```
<?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);
```

- Zusammenfassung:
  - `jdbcInbound` : Ein Jdbc-Kanaladapter. Es führt das SQL `SELECT * FROM BOOKS` und konvertiert die Ergebnismenge über den Bean `bookRowMapper` in `List<Book>` . Zum Schluss wird diese Buchliste an den Kanal `channel` .
  - `channel` : Übertragen Sie die Nachricht
  - `outbound` : ein generischer Outbound-Adapter. siehe Generic Inbound und Outbound Channel Adapter

Jdbc Outbound Channel Adapter - XML-Konfiguration

In der Referenzdokumentation zur Spring Integration heißt es:

> Der Outbound Channel Adapter ist die Umkehrung des Inbound: Seine Aufgabe ist es, eine Nachricht zu verarbeiten und sie zum Ausführen einer SQL-Abfrage zu verwenden. Die Nachrichten-Payload und -Header sind standardmäßig als Eingabeparameter für die Abfrage verfügbar.

- 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-Konfigurationsdatei

```
<?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
);
```

- Sie können die Tabelle `BOOKS` beobachten und sehen, dass die Datensätze eingefügt werden. Oder Sie können einen `int-jdbc:inbound-channel-adapter` schreiben, um die Tabelle `BOOKS` zu zählen, und Sie können feststellen, dass die Anzahl der Zähler kontinuierlich wächst.

- Zusammenfassung:

    - `inbound` : Ein generischer Inbound-Adapter, mit dem das `Book` Objekt als Nachrichtennutzlast abgerufen und an den Channel- `channel` .
    - `channel` : wird zum Übertragen der Nachricht verwendet.
    - `jdbcOutbound` : Ein jdbc-Outbound-Adapter. Er empfängt die Nachricht mit dem `Book` Typ und bereitet dann den Abfrageparameter vor `:title` und `:price` über das `sqlParameterSource` Bean mit SpEL wie `payload.title` und `payload.price` , um den Titel und den Preis aus der Nachrichtennutzlast `payload.price` .

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

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit der Spring-Integration	Community, Maxi Wu, walsh
2	Jdbc-Integration	walsh