EBook Gratuito

APPRENDIMENTO spring-integration

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-integration

Sommario

Di

Capitolo 1: Iniziare con l'integrazione di primavera

Osservazioni

Questa sezione fornisce una panoramica di cosa è l'integrazione di primavera e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare tutti i soggetti di grandi dimensioni all'interno dell'integrazione primaverile e collegarsi agli argomenti correlati. Poiché la Documentazione per l'integrazione di primavera è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Versioni

Versione	Data di rilascio
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

Installazione o configurazione

Il modo migliore per iniziare a utilizzare Spring-Integration nel tuo progetto è con un sistema di gestione delle dipendenze, come gradle.

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

Qui di seguito è un esempio molto semplice utilizzando i Gateway , servizio-attivatore endpoint messaggio.

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

Mostrerà la stringa `Hi, Spring Integration!` nella console.

Naturalmente, Spring Integration fornisce anche la configurazione in stile xml. Per l'esempio precedente, è possibile scrivere il seguente file di configurazione xml.

```
<?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>
```

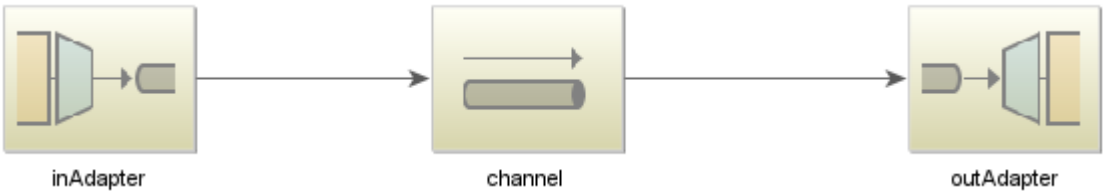
Per eseguire l'applicazione utilizzando il file di configurazione xml, è necessario modificare il codice `new AnnotationConfigApplicationContext(Application.class)` nella classe `Application` in `new ClassPathXmlApplicationContext("classpath:getstarted.xml")` . Ed esegui nuovamente questa applicazione, puoi vedere lo stesso output.

Adattatore canale in entrata e uscita generico

L'adattatore di canale è uno degli endpoint del messaggio in Spring Integration. È usato per il flusso di messaggi unidirezionale. Esistono due tipi di adattatori di canale:

Adattatore in entrata : lato di ingresso del canale. Ascolta o legge attivamente il messaggio.

Adattatore in uscita : lato di uscita del canale. Invia un messaggio alla classe Java o al sistema o protocollo esterno.



- Codice sorgente.

```
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

    }
}
```

- File di configurazione in stile XML:

```
<?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>
```

- Flusso di messaggi
    - `inAdapter` : un adattatore per canale in entrata. Richiama il metodo `Application$MessageProducer.produce` ogni 1 secondo (`<int:poller fixed-rate="1000"/>`) e invia la stringa restituita come messaggio al canale del `channel` .
    - `channel` : canale per trasferire il messaggio.
    - `outAdapter` : un adattatore per canale in uscita. Una volta raggiunto il messaggio sul canale `channel` , questo adattatore riceverà il messaggio e quindi lo invierà al metodo `Application$MessageConsumer.consume` che stampa il messaggio sulla console.
    - Quindi è possibile osservare che questa stringa di scelta casuale verrà visualizzata sulla console ogni 1 secondo.

Semplice Echo Excample con Spring-Integration-Stream

Codice Java:

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

File di configurazione Xml

```
<?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>
```

Questo è un esempio di eco. Quando si esegue questa applicazione Java, è possibile immettere una stringa e quindi verrà visualizzata sulla console.

Leggi Iniziare con l'integrazione di primavera online: https://riptutorial.com/it/spring-integration/topic/6985/iniziare-con-l-integrazione-di-primavera

Capitolo 2: Integrazione con Jdbc

Examples

Jdbc Inbound Adapter - configurazione xml

Nel documento di riferimento ufficiale , dice:

> La funzione principale di un adattatore di canale in entrata è eseguire una query SQL SELECT e trasformare il set di risultati in un messaggio. Il carico utile del messaggio è l'intero set di risultati, espresso come Elenco, e i tipi di elementi nell'elenco dipendono dalla strategia di mappatura delle righe utilizzata. La strategia predefinita è un mapper generico che restituisce solo una mappa per ogni riga nel risultato della query.

- Codice sorgente

```
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");
    }
}
```

- file di configurazione xml

```
<?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);
```

- Sommario:

  - `jdbcInbound` : un adattatore per canale in entrata Jdbc. Esegue SQL `SELECT * FROM BOOKS` e converte il set di risultati in `List<Book>` tramite il bean `bookRowMapper` . Infine, invia questo elenco di libri al canale del `channel` .
  - `channel` : trasferisci il messaggio
  - `outbound` : un adattatore in uscita generico. vedi Adattatore canale in entrata e uscita generico

Adattatore canale in uscita Jdbc - configurazione xml

Nel documento di riferimento sull'integrazione di primavera , dice:

> L'Adattatore canale in uscita è l'inverso dell'ingresso: il suo ruolo è quello di gestire un messaggio e utilizzarlo per eseguire una query SQL. Il carico utile del messaggio e le intestazioni sono disponibili per impostazione predefinita come parametri di input per la query ...

- Codice Java

```
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");
    }
}
```

- file di configurazione xml

```
<?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
);
```

- Puoi osservare la tabella `BOOKS` e puoi vedere i record inseriti. Oppure puoi scrivere un `int-jdbc:inbound-channel-adapter` in `int-jdbc:inbound-channel-adapter` per contare la tabella `BOOKS` e puoi scoprire che il numero di conteggio cresce in continuazione.

- Sommario:

    - `inbound` : un adattatore in entrata generico utilizzato per ottenere l'oggetto `Book` come payload del messaggio e inviarlo al canale del `channel` .
    - `channel` : usato per trasferire il messaggio.
    - `jdbcOutbound` : un adattatore in uscita JDBC, che riceve il messaggio con `Book` tipo e quindi preparare il parametro di query `:title` e `:price` via `sqlParameterSource` bean utilizzando SPEL come `payload.title` e `payload.price` per ottenere il titolo e prezzo di formare il messaggio payload.

Leggi Integrazione con Jdbc online: https://riptutorial.com/it/spring-integration/topic/8140/integrazione-con-jdbc

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con l'integrazione di primavera	Community, Maxi Wu, walsh
2	Integrazione con Jdbc	walsh