**Системная и программная инженерия**

Лабораторная работа №6 «Знакомство с Spring Integration»

**Введение**

Если смотреть на разработку программного обеспечения и бизнес-потребности с точки зрения разработчика или архитектора, то речь всегда идет об интеграции компонентов и систем, либо внутренних по отношению к нашей архитектуре, либо внешних, и выяснении, какие варианты обеспечивают полную функциональность, высокую доступность и легко поддаются сопровождению и расширению.

**Spring Integration** — простая модель реализации корпоративных решений интеграции. С ее помощью можно сделать приложение Spring Boot асинхронным и ориентированным на обработку сообщений. Spring Integration благодаря реализации всех корпоративных паттернов интеграции позволяет создавать корпоративные, надежные, переносимые интегрированные решения.

Основные компоненты Spring Integration таковы:

***Сообщение* (message).** Универсальная обертка для любого Java-объекта. Состоит из заголовков и содержимого. Заголовки обычно содержат важную информацию, например идентификатор, метку даты/времени, идентифи катор корреляции и обратный адрес, и конечно, вы можете добавлять свои собственные поля. Содержимое может представлять собой данные произ вольного типа, от байтовых массивов до пользовательских объектов.

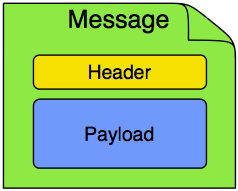


Рис. 1 – Сообщение Spring Integration

***Канал передачи сообщений* (message channel).** Такой канал передачи сообщений следует паттернам обмена сообщениями, например, паттерну «точка — точка» и модели «публикация/подписка». В Spring Integration есть несколько каналов передачи сообщений, в частности каналы передачи сообщений с опросом (благодаря которым возможна буферизация сообщений в очереди) или каналы передачи сообщений с подпиской̆, для потребителей̆.

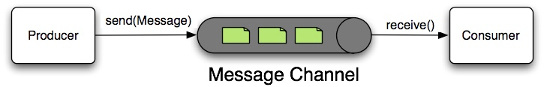


Рис. 2 –  Канал передачи сообщений Spring Integration

***Конечная точка сообщений* (Message endpoint).** Фильтр, соединяющий код приложения с фреймворком обмена сообщениями. Большинство этих конечных точек входят в число реализаций из *Enterprise Integration Patterns.*

***Фильтр*(Filter).**Фильтр сообщений определяет, когда сообщение следует передавать в выходной канал.

***Модификатор* (Transformer)**. Модификатор сообщений модифицирует содержимое или структуру сообщения и передает его в выходной канал.

***Маршрутизатор*(Router).** Маршрутизатор сообщений определяет на основе заданных правил, что делать и куда отправлять сообщение. Эти правила могут содержаться в заголовках или в содержимом сообщения.

***Разделитель* (Splitter).** Разделитель сообщений получает сообщение (по входному каналу), разбивает его и возвращает несколько новых со общений (по выходному каналу).

***Активатор сервиса* (Service activator).** Конечная точка, играющая роль сервиса: получающая (по входному каналу) и обрабатывающая сообщение. Может завершать поток интеграции, возвращать то же сообщение или возвращать совершенно новое (по выходному каналу).

***Агрегатор*(Aggregator).**Эта конечная точка сообщений получает несколько сообщений (по входному каналу), объединяет их в одно новое сообщение (на основе стратегии выпуска) и выдает его наружу (по выходному каналу).

***Адаптеры каналов* (Channel adapters)**. Конечная точка, соединяющая канал передачи сообщений с другими системами или транспортными протоколами. Spring Integration предоставляет как входные, так и выходные адаптеры. На случай, когда требуется ответ, в нем есть адаптер-шлюз. Они используются чаще всего. Почему? Если вашему приложению нужно подключиться к RabbitMQ, JMS, FTP, файловой системе, HTTP или любой другой технологии, программировать клиент не нужно, в Spring Integration всегда найдется адаптер для подключения к ней.

**Практическая часть**

В качестве примера будем подключаться к Redis, используя механизмы Spring Integration.Перед началом работы убедитесь, что Redis запущен на вашей машине. (Можно запустить контейнер с образом редиса, как мы это делали в 2 лабораторной работе).

Пример для данной лабораторной работы можно посмотреть [на gitflic](https://gitflic.ru/project/aleks777k2/laboratornaya-rabota-6)

Необходимо создать новый maven-проект с помощью Spring Initializr:

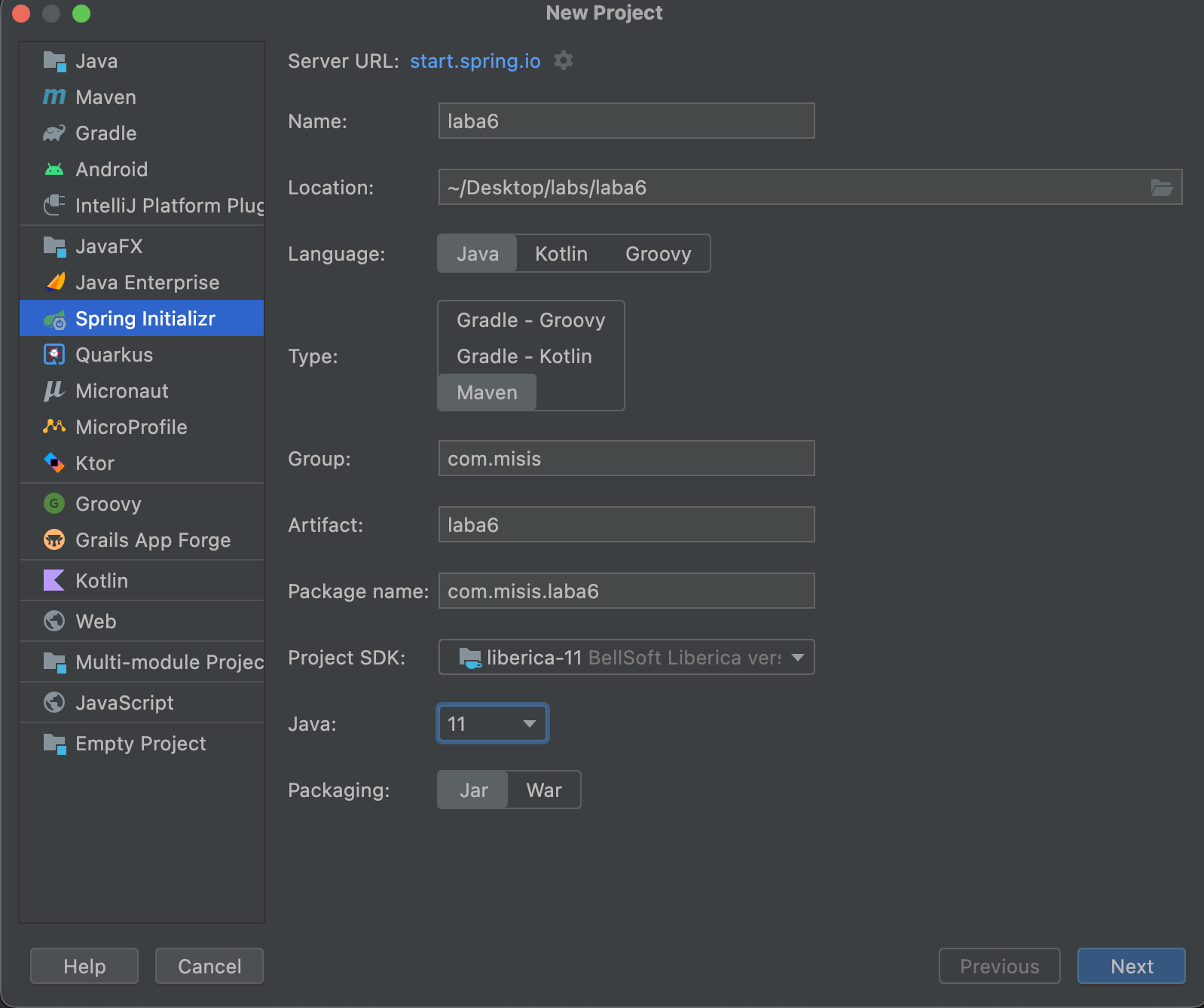


Рис. 3 – Создание spring boot приложения

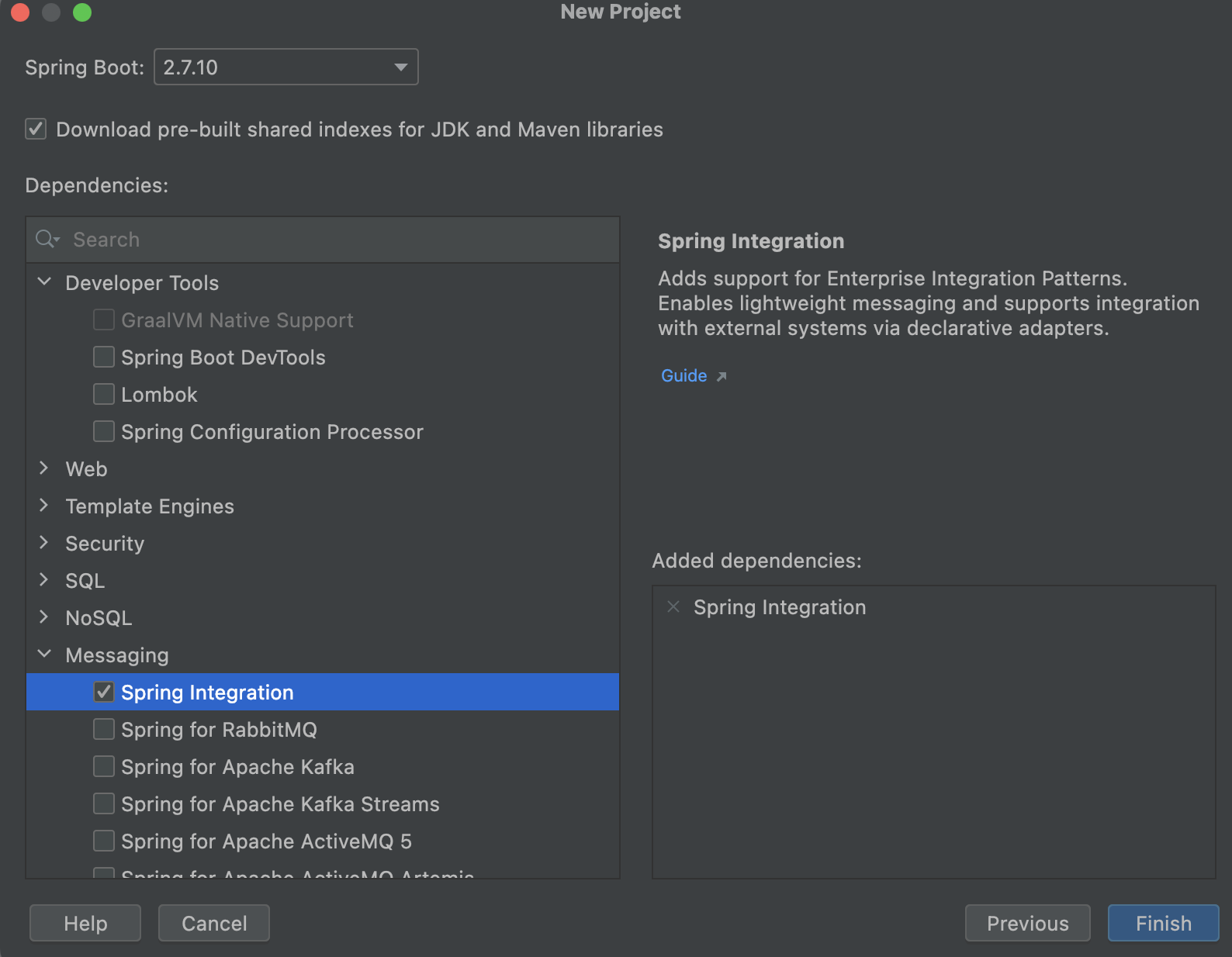


Рис. 4 – Создание spring boot приложения

Создадим Dto-объект, который мы будем помещать в payload сообщения Spring Integration:

package com.misis.laba6.dto;  
  
public class HelloWorldDto {  
 private String message;  
  
 public String getMessage() {  
 return message;  
 }  
  
 public HelloWorldDto setMessage(String message) {  
 this.message = message;  
 return this;  
 }  
}

Далее создадим необходимые конфигурационные Bean’ы, в которых определим InoundChannel, OutboundChannel и InoundChannelFlow, OutboundChannelFlow.

RedisConnectionFactory –  потоко-безопасная фабрика для соединений с Redis.

Jackson2JsonRedisSerializer – нужен для сериализации/десериализации наших java-объектов в Json.

package com.misis.laba6.configuration;  
  
import com.misis.laba6.dto.HelloWorldDto;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.data.redis.connection.RedisConnectionFactory;  
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;  
import org.springframework.integration.channel.PublishSubscribeChannel;  
import org.springframework.integration.config.EnableIntegration;  
import org.springframework.integration.dsl.IntegrationFlow;  
import org.springframework.integration.dsl.IntegrationFlows;  
import org.springframework.integration.redis.inbound.RedisQueueMessageDrivenEndpoint;  
import org.springframework.integration.redis.outbound.RedisQueueOutboundChannelAdapter;  
import org.springframework.messaging.MessageChannel;  
import org.springframework.messaging.SubscribableChannel;  
  
  
@Configuration  
@EnableIntegration  
public class RedisIntegrationConfiguration {  
 private static final String *QUEUE\_HELLO\_WORLD* = "queue:hello-world";  
  
 @Bean("helloWorldInboundChannelFlow")  
 public IntegrationFlow redisHelloWorldEventInboundChannelFlow(  
 RedisConnectionFactory redisConnectionFactory,  
 @Qualifier("helloWorldInboundChannel") MessageChannel channel  
 ) {  
 RedisQueueMessageDrivenEndpoint endpoint =  
 new RedisQueueMessageDrivenEndpoint(*QUEUE\_HELLO\_WORLD*, redisConnectionFactory);  
 Jackson2JsonRedisSerializer<HelloWorldDto> serializer  
 = new Jackson2JsonRedisSerializer<>(HelloWorldDto.class);  
  
 endpoint.setSerializer(serializer);  
 endpoint.setBeanName("helloWorldRedisQueueMessageDrivenEndpoint");  
  
 return IntegrationFlows  
 .*from*(endpoint)  
 .channel(channel)  
 .get();  
 }  
  
 @Bean("helloWorldOutboundChannelFlow")  
 public IntegrationFlow redisHelloWorldEventOutboundChannelFlow(  
 RedisConnectionFactory redisConnectionFactory,  
 @Qualifier("helloWorldOutboundChannel") MessageChannel channel  
 ) {  
 Jackson2JsonRedisSerializer<HelloWorldDto> serializer  
 = new Jackson2JsonRedisSerializer<>(HelloWorldDto.class);  
  
 RedisQueueOutboundChannelAdapter channelAdapter =  
 new RedisQueueOutboundChannelAdapter(*QUEUE\_HELLO\_WORLD*, redisConnectionFactory);  
 channelAdapter.setSerializer(serializer);  
 return IntegrationFlows  
 .*from*(channel)  
 .handle(channelAdapter)  
 .get();  
 }  
  
 @Bean("helloWorldOutboundChannel")  
 public SubscribableChannel logEventOutboundChannel() {  
 PublishSubscribeChannel channel = new PublishSubscribeChannel();  
 channel.setMaxSubscribers(3);  
 channel.setComponentName("helloWorldOutboundChannel");  
  
 return channel;  
 }  
  
 @Bean("helloWorldInboundChannel")  
 public SubscribableChannel logEventInboundChannel() {  
 PublishSubscribeChannel channel = new PublishSubscribeChannel();  
 channel.setMaxSubscribers(3);  
 channel.setComponentName("helloWorldInboundChannel");  
 return channel;  
 }  
  
}

Для того, чтобы прошло корректное подключение к Redis необходимо указать следующие параметры в application.properties файле:

spring.redis.host=localhost

spring.redis.port=6379 (если Редис запущен на другом порту, то следует изменить этот параметр)

Для передачи сообщений в OutboundChannel определим Gateway:

@Component  
public class HelloWorldGateway {  
  
 private final MessageChannel messageChannel;  
  
 @Autowired  
 public HelloWorldGateway(  
 @Qualifier("helloWorldOutboundChannel") MessageChannel messageChannel  
 ) {  
 this.messageChannel = messageChannel;  
 }  
  
 public void send(HelloWorldDto event) {  
 messageChannel.send(MessageBuilder.*withPayload*(event).build());  
 }  
  
}

Для того чтобы мы могли получать сообщения необходимо определить активатор:

package com.misis.laba6.activator;  
  
import com.misis.laba6.dto.HelloWorldDto;  
import org.springframework.integration.annotation.ServiceActivator;  
import org.springframework.messaging.Message;  
import org.springframework.stereotype.Component;  
  
@Component  
public class HelloWorldActivator {  
  
  
 @ServiceActivator(inputChannel = "helloWorldInboundChannel")  
 public void activate(Message<HelloWorldDto> event) {  
  
 System.*out*.println(event.getPayload().getMessage());  
 }  
}

Все необходимые программные компоненты были сконфигурированы, теперь можно запускать наше приложение:

package com.misis.laba6.component;  
  
import com.misis.laba6.dto.HelloWorldDto;  
import com.misis.laba6.gateway.HelloWorldGateway;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.stereotype.Component;  
  
@Component  
public class ComandLineComponent implements CommandLineRunner {  
 private final HelloWorldGateway helloWorldGateway;  
  
 @Autowired  
 public ComandLineComponent(HelloWorldGateway helloWorldGateway) {  
 this.helloWorldGateway = helloWorldGateway;  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 HelloWorldDto dto = new HelloWorldDto();  
 dto.setMessage("HELLO WORLD!!!!!!!!!!!!");  
 helloWorldGateway.send(dto);  
  
 }  
}

В результате в консоли мы получим:

Изображение выглядит как текст

Автоматически созданное описание

Рис. 5 – Результат отправки

**Практическое задание**

1. Необходимо создать с помощью Spring Integration небольшое приложение для обмена сообщениями: В выходящий канал передается объект, содержащий следующие поля: fullName(ФИО студента); birthday. А в активаторе, получающего сообщение из входного канала, необходимо вывести содержимое(payload) полученного сообщения.

1. Для объекта из практического задания 1 необходимо сделать новое поле randomValue. В выходящий канал отправить 10 объектов с случайными значениями (от 1 до 100) randomValue. С помощью [Фильтра](https://docs.spring.io/spring-integration/api/org/springframework/integration/annotation/Filter.html) отправлять в входной канал лишь те сообщения, значение randomValue которых нечетно.