ИНТЕГРАЦИЯ МИКРОСЕРВИСОВ

Владимир Плизга́
ЦФТ

ТЫ ВООБЩЕ КТО?

- Владимир Плизгá
- Backend разработчик
  - **Позиция**: инженер-программист
  - **Область**: разработка Интернет-банков и сопутствующих сервисов
  - **Опыт** в области: 6+ лет

https://oplata.kykyryza.ru

- 20+ партнёров по Интернет-банкам
- 20+ партнёров по сайтам подарочных карт

В России и СНГ

ЧТО ТАКОЕ SPRING INTEGRATION?

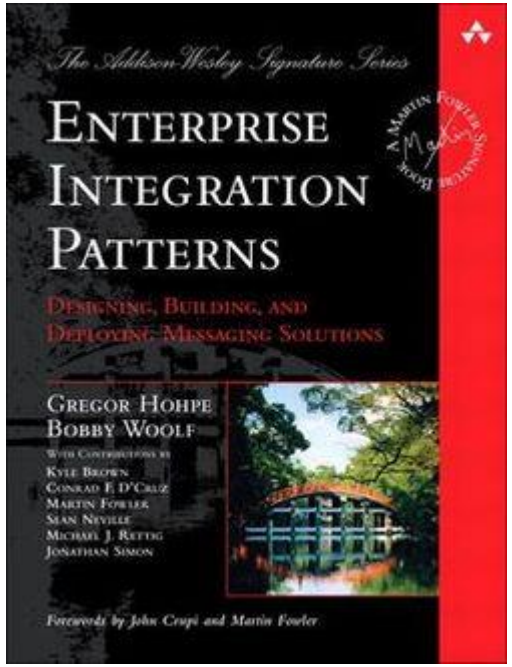
- Проект в экосистеме Spring

Spring Integration = Spring IoC + EIP

* **IoC** – Inversion of Control
* **EIP** – Enterprise Integration Patterns

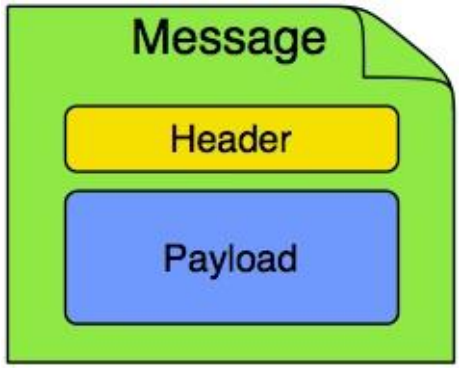
ВКРАТЦЕ О EIP

- Программная модель для построения интеграций
  - Как внутри приложения
  - Так и между приложениями
- 65 паттернов на все случаи жизни
- Обеспечивает слабую связность компонентов
- Не зависит от стека технологий



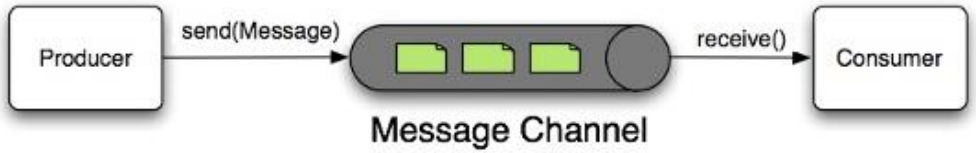
КОНЦЕПЦИИ EIP: MESSAGE & CHANNEL

Сообщение




```
public interface Message<T> {
  T getPayload();
  MessageHeaders getHeaders();
}
```

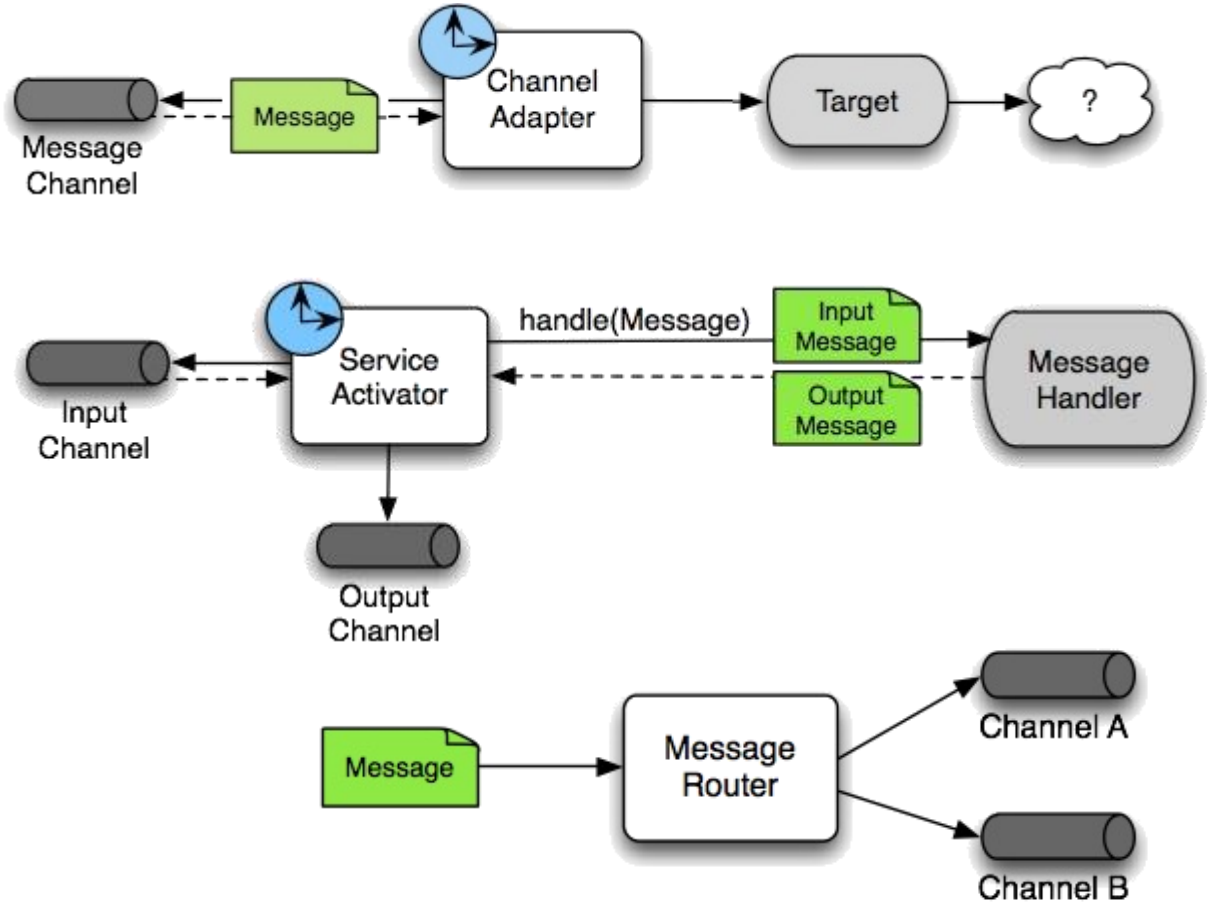
Канал




```
@FunctionalInterface
public interface MessageChannel {
  boolean send(Message<?> message);
}
```

КОНЦЕПЦИИ EIP: ENDPOINT

- Transform
- Filter
- Handle
- Split
- Aggregate
- Route
- Bridge
- …



КОНЦЕПЦИИ EIP: ADAPTER

- AMQP/JMS
- JBDC/JPA/NoSQL
- File
- Mail
- Social
- FTP[S]/SFTP
- HTTP/WebFlux/WebSocket
- RMI
- …

ПРИЧЁМ ТУТ SPRING INTEGRATION?

«Спецификация»	«Имплементация»
JVMS	HotSpot, JRockit, Excelsior JET
JPA	Hibernate
AMQP	RabbitMQ, ActiveMQ
EIP	Apache Camel, **Spring Integration**

Набор реализующих EIP компонентов, позволяющих разработчику концентрироваться на бизнес-логике и минимально заниматься инфраструктурой обмена сообщениями

КАК РАБОТАЕТ SPRING INTEGRATION

- Комбинация каналов и точек обработки (endpoints) – конвейер (flow)
- Все элементы конвейера – бины в Spring-контексте
- Конвейером может быть как всё приложение, так и любые участки
- 3 способа задания конфигурации конвейеров (не исключающие):
  1. XML
  2. Java Annotations
  3. DSLs: Java, Groovy, Scala

CONFIGURATION: XML

```
<int:gateway id="cafe" service-interface="org.springframework.integration.samples.cafe.Cafe"/>

<!-- each order has a collection of order items that is split apart to be processed -->
<int:channel id="orders"/>
<int:splitter input-channel="orders" expression="payload.items" output-channel="drinks"/>

<!-- drink order items are aggregated in a call to the waiter -->
<int:channel id="preparedDrinks"/>
<int:aggregator input-channel="preparedDrinks"  method="prepareDelivery"
         output-channel="deliveries">
   <beans:bean class="org.springframework.integration.samples.cafe.xml.Waiter"/>
</int:aggregator>

<int-stream:stdout-channel-adapter id="deliveries"/>
<beans:bean id="barista" class="org.springframework.integration.samples.cafe.x
<int:poller id="poller" default="true" fixed-delay="1000"/>
```

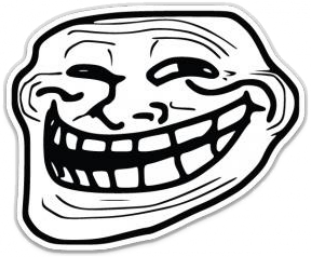
CONFIGURATION: JAVA ANNOTATION

```
@Bean
@Transformer(inputChannel = "enrichHeadersChannel", outputChannel = "emailChannel")
public HeaderEnricher enrichHeaders() {
    Map<String, ? extends HeaderValueMessageProcessor<?>> headersToAdd =
            Collections.singletonMap("emailUrl",
                    new StaticHeaderValueMessageProcessor<>(this.imapUrl));
    HeaderEnricher enricher = new HeaderEnricher(headersToAdd);
    return enricher;
}
```

```
@Bean
@Transformer(inputChannel="enrichHeadersChannel", outputChannel="emailChannel")
public HeaderEnricher enrichHeaders() {
    Map<String, HeaderValueMessageProcessor<?>> headersToAdd = new HashMap<>();
    headersToAdd.put("emailUrl", new StaticHeaderValueMessageProcessor<String>(this.imapUrl));
    Expression expression = new SpelExpressionParser().parseExpression("payload.from[0].toString()");
    headersToAdd.put("from",
                new ExpressionEvaluatingHeaderValueMessageProcessor<>(expression, String.class));
    HeaderEnricher enricher = new HeaderEnricher(headersToAdd);
    return enricher;
}
```

CONFIGURATION: JAVA DSL

- Основан на паттерне Builder и Java Annotation Configuration
- *«Another useful tool to simplify configuration is Java 8 Lambdas.»*
- Всем видам endpoint'ов соответствуют методы builder'а
- Каждый конвейер – экземпляр класса **IntegrationFlow,** регистрируемый в виде Spring бина:



```
@Bean
public IntegrationFlow integerFlow() {
    return IntegrationFlows.from("input")    // input – имя канала
            .<String, Integer> transform(Integer::parseInt)
            .log()
            .get();
}
```

implicit direct channel

БЛИЖЕ К ДЕЛУ

Практическая часть

БИЗНЕС ТРЕБОВАНИЕ

Требуется:

- Уведомлять всех желающих пользователей о происходящих событиях в матчах Чемпионата мира 2018 посредством сообщений Telegram.

ДЕКОМПОЗИЦИЯ ЗАДАЧИ

1. Периодические запросы событий матча
2. Рассылка сообщений пользователям

Инструментарий:
Spring Boot + Spring Integration

Let’s have some code!

ПРОТИВОПОКАЗАНИЯ

- Когда легко обойтись обычным Java Stream API
- Когда в команде мало носителей знаний по Spring Integration
- Когда нужен конвейер для массивной транзакционной обработки (для этого есть Spring Batch)

ОСТАЛОСЬ ЗА КАДРОМ

- Каналы: Queue, Executor, Priority, …
- Endpoints: Filter, Aggregator, Gateway, …
- Message & Metadata Store
- Мониторинг (в т.ч. JMX)
- Управление транзакциями
- Обработка ошибок
- Тестирование

https://github.com/Toparvion/footbot

HAPPY INTEGRATION!

Владимир Плизгá (ЦФТ)
toparvion@gmx.com