Spring Integration

3대 메인 컴포넌트

Message

- Header + Payload

Message Channel

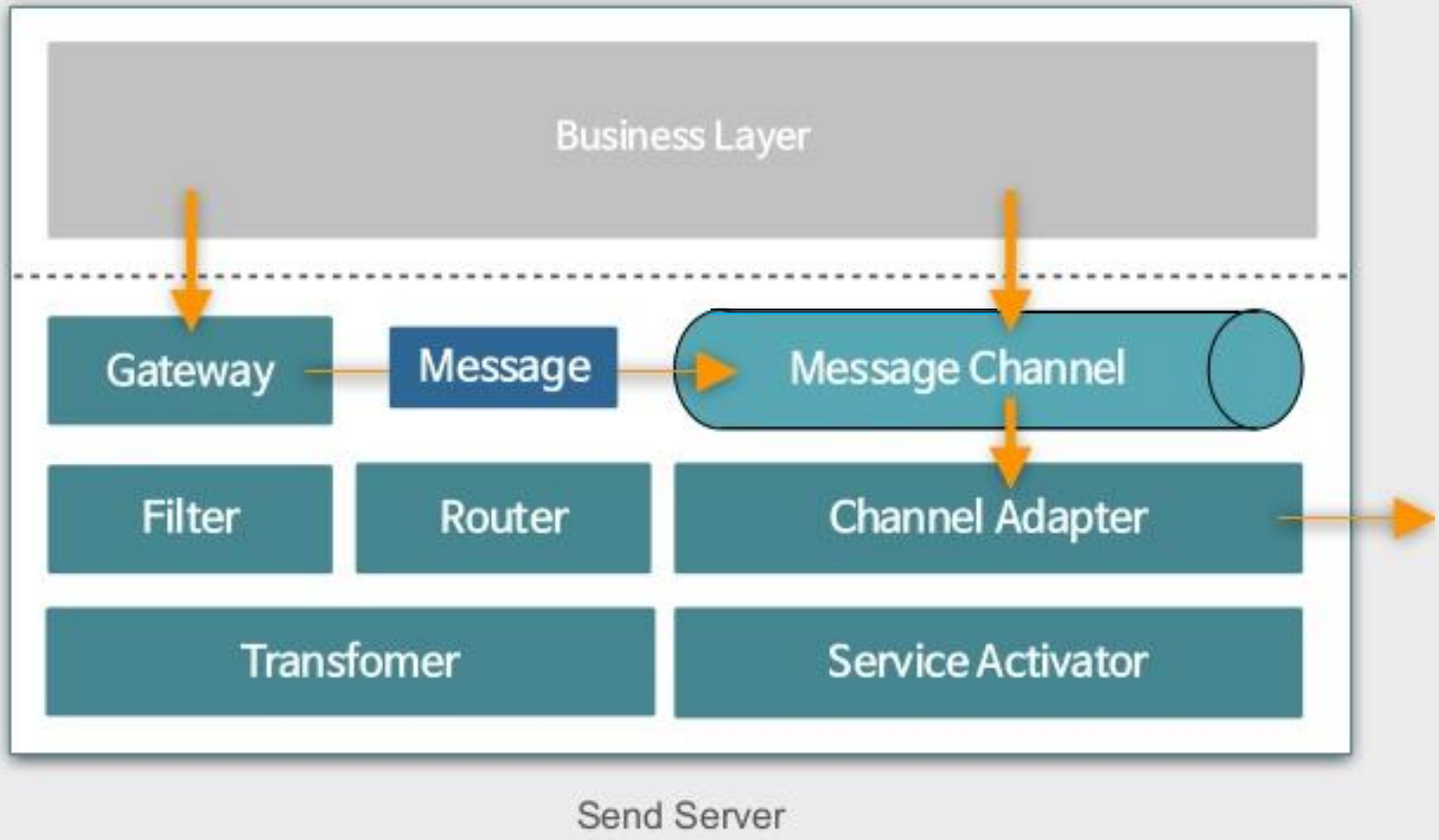
- pipes-and-filters 구조에서 **pipe** 의 역할
- Producer -> Message Channel -> Consumer 구조
- Spring Integration 의 Message Channel은 Point-to-Point 또는 Publish/Subscribe 방식을 제공

Message Endpoint

- pipes-and-filters 구조에서 **filter** 의 역할
- 엔드포인트의 주요 역할은 어플리케이션 코드를 메시징 프레임웍에 연결
- 엔드포인트의 역할은 MVC 패턴에서 컨트롤러의 역할과 비슷

Spring Integration

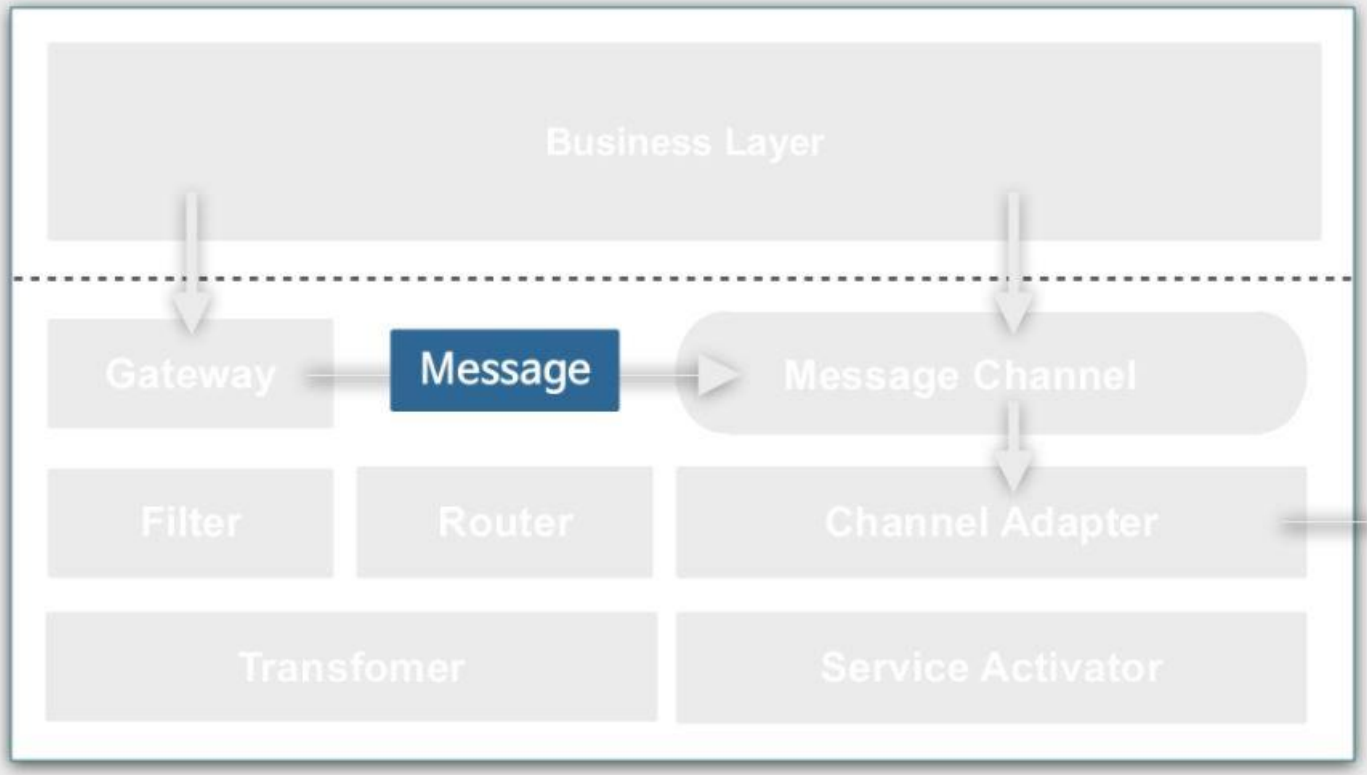
Spring Integration의 3대 메인 컴포넌트



Spring Integration

Message is

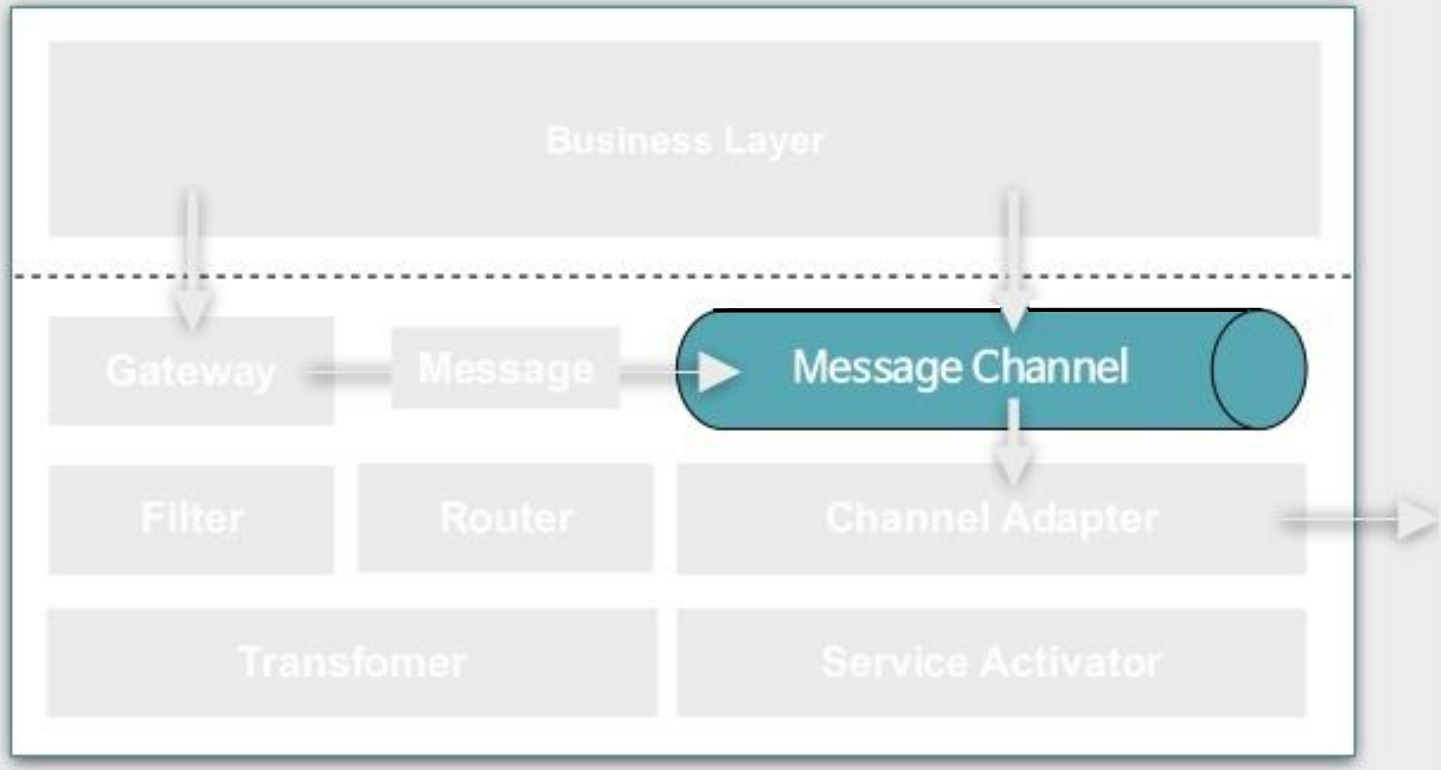
목적지로 발신 / 수신할 데이터의 Wrapper Class



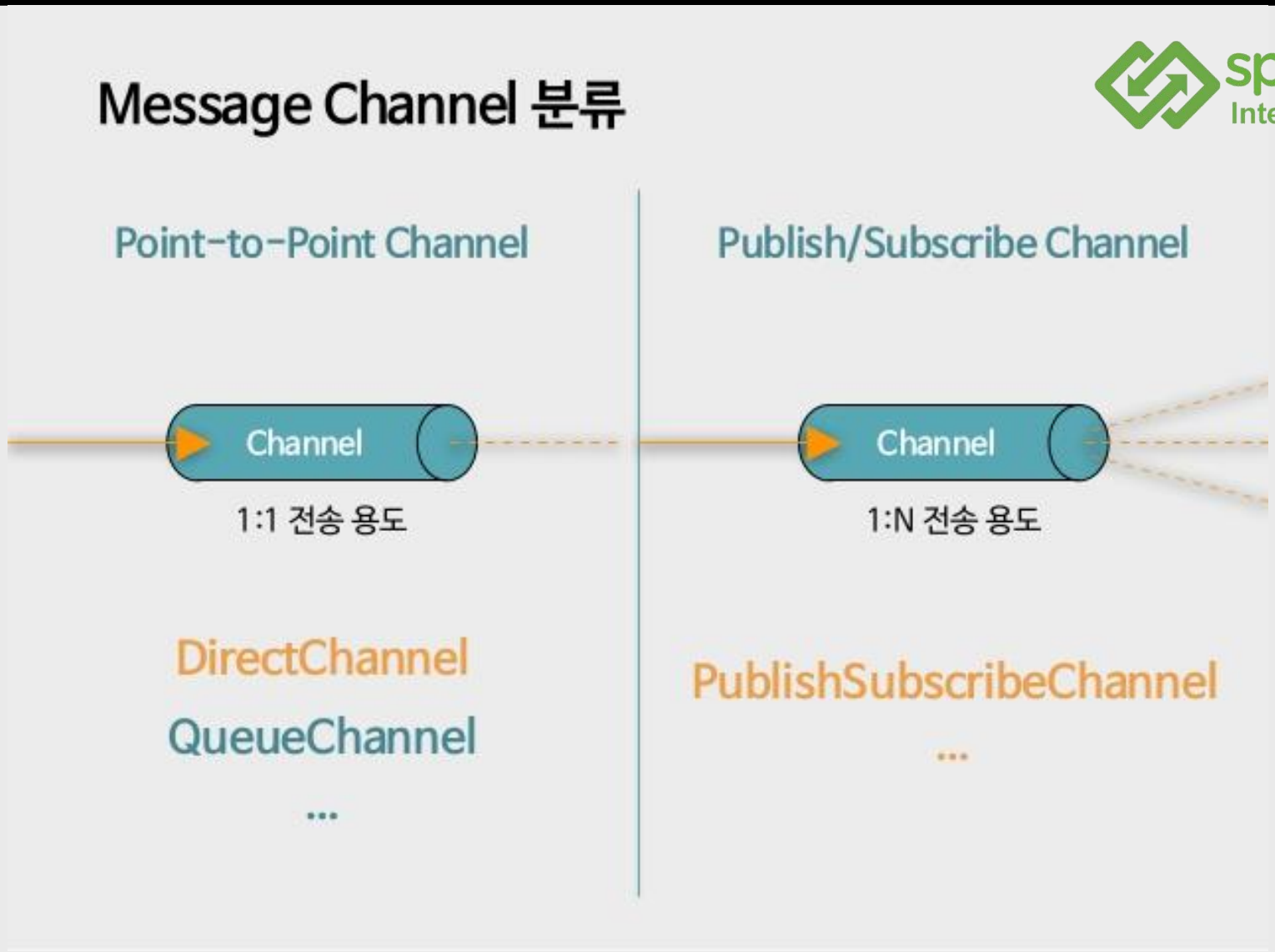
Spring Integration

Message Channel is

비지니스 로직과 엔드포인 사이에서 메시지를 보내거나 받는 통로 역할



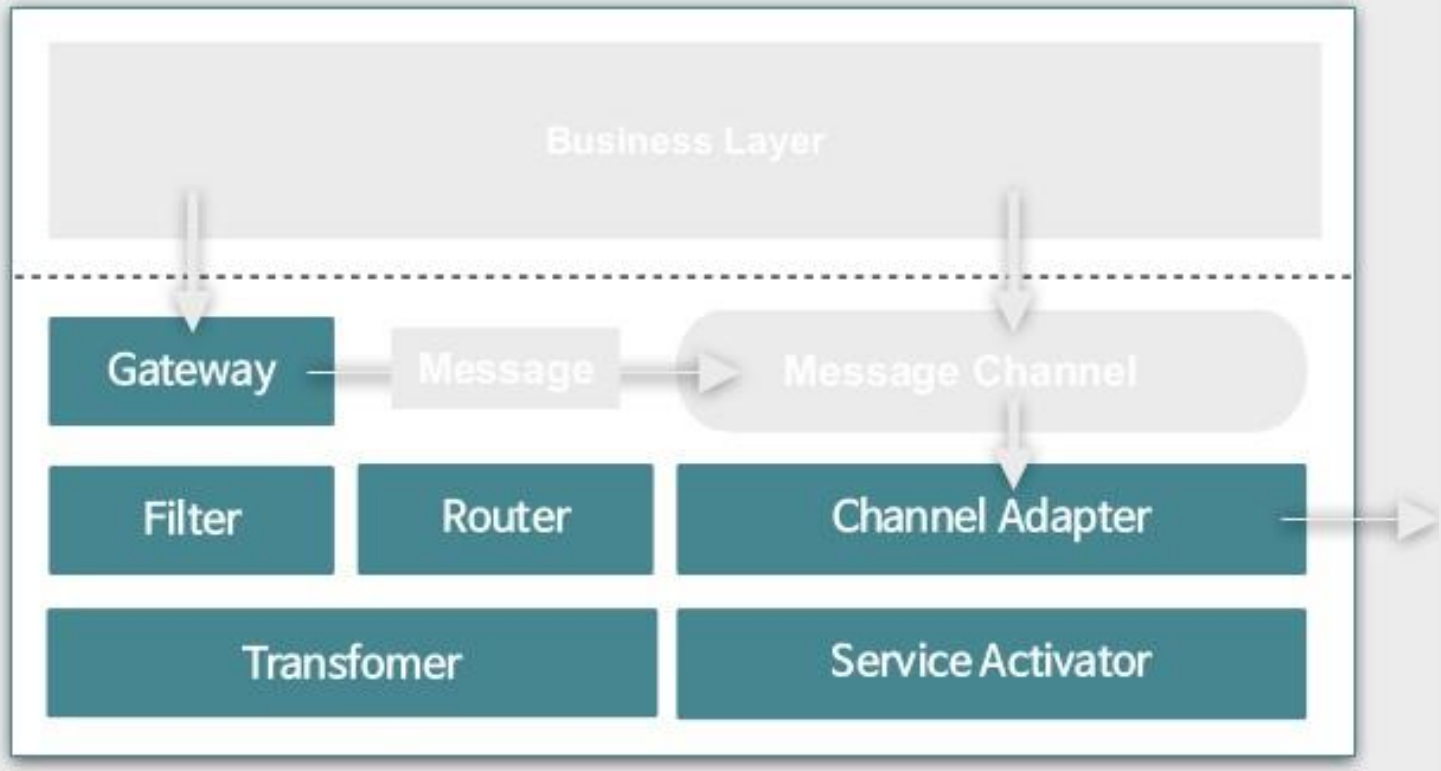
Spring Integration



Spring Integration

Message Endpoints is

메시지 채널을 통해 메시지 송수신 등과 같은 처리를 돕는 모듈



Message Endpoint

Gateway	비즈니스 로직에서 메시지 송수신을 쉽게 도와주는 컴포넌트
Service Activator	메시지가 입력 채널에 도착했을 때 특정 빈의 메소드를 호출해주는 컴포넌트
Channel Adapter	외부 시스템과 메시지 송수신하기 위한 컴포넌트
Transformer	입력 채널로 들어온 페이로드를 특정 목적에 맞게 변화하여 출력 채널로 전달하는 엔드포인트
Filter	특정 채널로 어떤 메시지는 전달하지 말아야 할지 걸러내는 컴포넌트
Router	메시지를 한 개 이상의 채널로 보내는 컴포넌트

TcpClientConfig


```
@Bean
public AbstractClientConnectionFactory clientConnectionFactory() {
  List<AbstractClientConnectionFactory> clientFactories = new ArrayList<AbstractClientConnectionFactory>();
  for (String host_ : host) {
    for (String port_ : port) {
      int p = Integer.valueOf(port_.trim());

1     TcpNioClientConnectionFactory tcpNioClientConnectionFactory = new TcpNioClientConnectionFactory(host_, p);
      CustomByteArrayLengthHeaderSerializer serializer = new CustomByteArrayLengthHeaderSerializer();
      tcpNioClientConnectionFactory.setSerializer(serializer);
      tcpNioClientConnectionFactory.setDeserializer(serializer);
      tcpNioClientConnectionFactory.setUsingDirectBuffers(true);
      tcpNioClientConnectionFactory.setApplicationEventPublisher(applicationEventPublisher);
      tcpNioClientConnectionFactory.setSingleUse(SOCKET_SINGLE_USE);

2     CachingClientConnectionFactory cachingClient = new
            CachingClientConnectionFactory(tcpNioClientConnectionFactory, connectionPoolSize);
      cachingClient.setBeanName(String.format("cache_%s", host_));
      cachingClient.setSingleUse(SOCKET_SINGLE_USE);
      cachingClient.afterPropertiesSet();

      clientFactories.add(cachingClient);
    }
  }
3 FailoverClientConnectionFactory failoverClient = new FailoverClientConnectionFactory(clientFactories);
  failoverClient.setSingleUse(SOCKET_SINGLE_USE);
  failoverClient.afterPropertiesSet();

  return failoverClient;
}
```

TcpClientConfig

4

```
@Bean
public MessageChannel outboundChannel() {
  return new DirectChannel();
}
```

5

```
@Bean
@ServiceActivator(inputChannel = "outboundChannel")
public MessageHandler outboundGateway(AbstractClientConnectionFactory clientConnectionFactory) {
  TcpOutboundGateway tcpOutboundGateway = new TcpOutboundGateway();
  tcpOutboundGateway.setConnectionFactory(clientConnectionFactory);
  tcpOutboundGateway.setRequestTimeout(TIMEOUT);
  tcpOutboundGateway.setRemoteTimeout(TIMEOUT);
  return tcpOutboundGateway;
}
```

TcpServerConfig


```
@Bean
public AbstractServerConnectionFactory serverConnectionFactory() {
  TcpNioServerConnectionFactory tcpNioServerConnectionFactory = new TcpNioServerConnectionFactory(port);
  CustomByteArrayLengthHeaderSerializer serializer = new CustomByteArrayLengthHeaderSerializer();
  tcpNioServerConnectionFactory.setSerializer(serializer);
  tcpNioServerConnectionFactory.setDeserializer(serializer);
  tcpNioServerConnectionFactory.setUsingDirectBuffers(true);
  return tcpNioServerConnectionFactory;
}

@Bean
public MessageChannel inboundChannel() {
  return new DirectChannel();
}

@Bean
public TcpInboundGateway inboundGateway(AbstractServerConnectionFactory serverConnectionFactory,
                                        MessageChannel inboundChannel) {
  TcpInboundGateway tcpInboundGateway = new TcpInboundGateway();
  tcpInboundGateway.setConnectionFactory(serverConnectionFactory);
  tcpInboundGateway.setRequestChannel(inboundChannel);
  return tcpInboundGateway;
}
```

1
2
3