

10장 메시징

애플리케이션에서 발생한 이벤트를 다른 프로세스와 네트워크에서 실행중인 서비스에 전달함으로써 애플리케이션을 다른 서비스와 연결하기 위한 방법

- 이벤트 알림 (event **notification**)

도메인 안에서 발생한 시스템 변경 사항을 이벤트로 다른 시스템에 전달
메시지를 받은 시스템은 송신 시스템에 응답 의무 없음
생성 이후 데이터 변경이 발생하면 안됨. 불변
이벤트 메시지에는 일부 ID 정보와 자세한 정보를 질의할 수 있는 링크

- 이벤트를 통한 상태 전송 (event-carried **state transfer**)

수신 시스템에서 송신 시스템에 별도의 콜백을 사용해서 데이터를 (추가로) 요청할 필요가 없는 메시지
이벤트
이벤트 메시지에는 수신 시스템이 이벤트를 처리할 때 필요한 모든 정보가 포함

- 이벤트 소싱 (event **sourcing**)

시스템의 상태를 변경할 때마다 해당 상태 변경을 이벤트로 기록
언제든지 이벤트를 재처리하여 시스템 상태를 재 구축 할 수 있음

메시지 브로커

서비스간 메시지를 저장/전달하기 위한 일종의 허브

- 아파치 카프카
- 래빗 엠큐
- 액티브 엠큐
- 엠큐시리즈

메시지를 보내는 프로듀서와 메시지를 처리하는 컨슈머는 서로 직접 연결 x
메시지 브로커에 접속

스프링 인티그레이션을 사용한 이벤트 주도 아키텍처

다양한 이벤트 형태

다양한 형태의 시스템

점대점으로 연결하기 쉽지 않음

스프링 인티그레이션 프레임워크

`Message<T>`

헤더(메타데이터) + 페이로드

`MessageChannel`

비즈니스 로직과 엔드포인트 사이에서 메시지를 주고받는 통로

`Message<T>` 객체 전달

Messaging Endpoints

메시지 채널을 통해 메시지 송수신등과 같은 처리를 돕는 컴포넌트

<https://www.slideshare.net/WangeunLee/spring-integration-47185594>

엔터프라이즈 애플리케이션 인티그레이션 패턴

필터, 라우터, 트랜스포머, 어댑터, 게이트웨이

메시징 종단점 (Messaging Endpoints)

메시지 채널을 통해 메시지 송수신등과 같은 처리를 돕는 컴포넌트

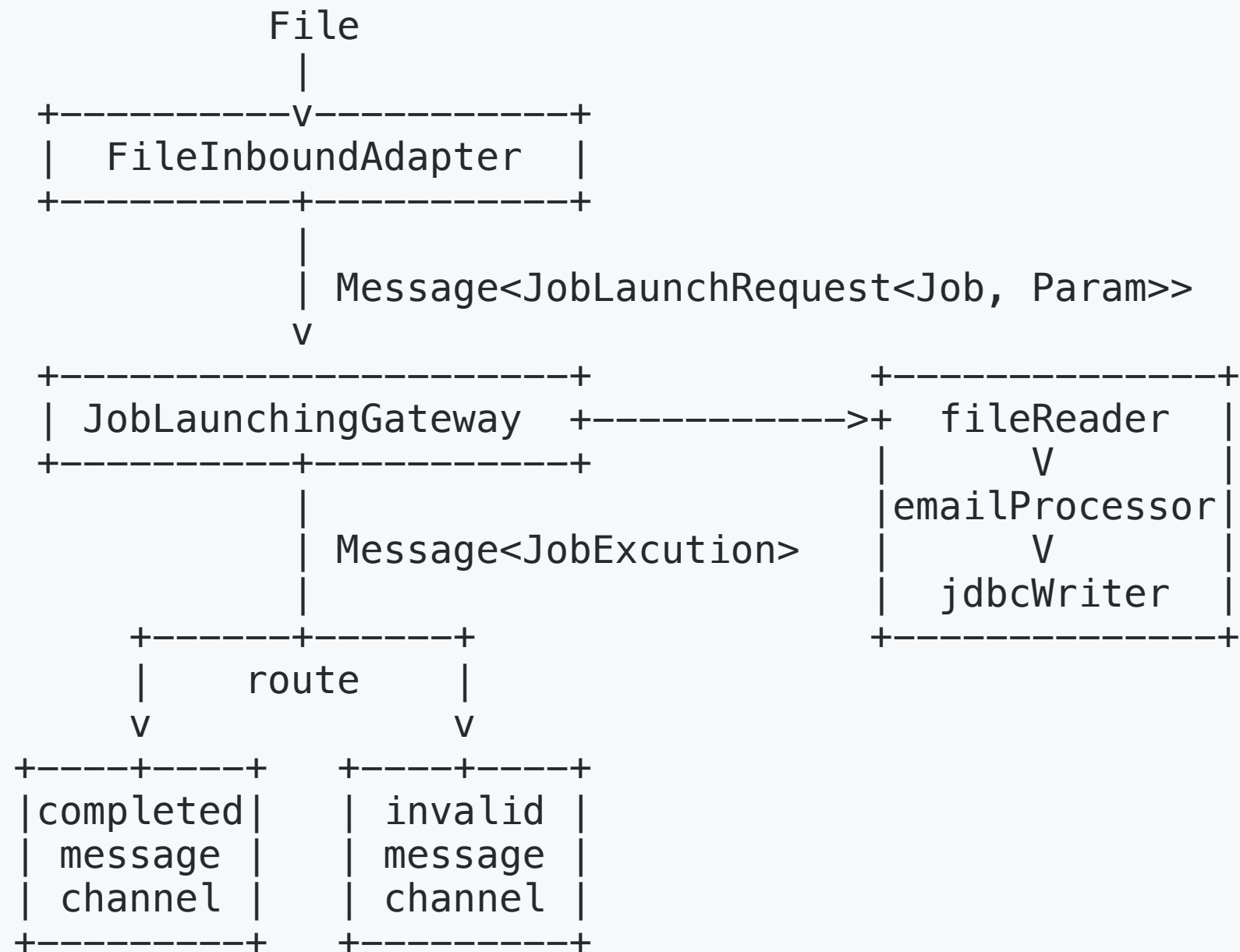
- 인바운드 게이트웨이: 외부 시스템으로부터 전송된 요청을 받아 `Message<T>` 로 처리해서 응답
- 아웃바운드 게이트웨이: `Message<T>` 를 받아 외부 시스템으로 전달하고 응답을 기다림
- 인바운드 어댑터: 외부로부터 받은 메시지를 스프링 `Message<T>` 로 변환
- 아웃바운드 어댑터: 스프링 `Message<T>` 를 받아 외부 다운스트림 시스템이 원하는 형태로 변환
- 필터: 유입되는 메시지 필터
- 라우터: 유입되는 메시지를 어느 다운스트림으로 전달할지 결정
- 트랜스포머: 메시지 변경
- 스플리터: 메시지를 더 작은 메시지로 나누어 다운스트림에 전달
- 애그리게이터: 여러개의 메시지를 받아 특정 속성에 관계가 있는 내용들로 모아서 다운스트림에 전달

예제 10-2

```
@Bean
fun etlFlow(@Value("\${input-directory:\${HOME}}") dir: File): IntegrationFlow =
    IntegrationFlows.from(Files.inboundAdapter(dir)) {
        it.poller { spec -> spec.fixedRate(1000) }
    }.handle(File::class.java) { file, _ ->
        log.info("we noticed a new file, $file")
        file
    }.routeToRecipients { spec ->
        spec.recipient(csv(), MessageSelector { msg -> hasExt(msg.payload, ".csv")
            .recipient(txt(), MessageSelector { msg -> hasExt(msg.payload, ".txt")
        }.get()

@Bean fun txt(): MessageChannel = MessageChannels.direct().get()
@Bean fun csv(): MessageChannel = MessageChannels.direct().get()
@Bean fun txtFlow(): IntegrationFlow = IntegrationFlows.from(txt()).handle(File:
    log.info("file is .txt!")
    null
}.get()
@Bean fun csvFlow(): IntegrationFlow = IntegrationFlows.from(csv()).handle(File:
    log.info("file is .csv!")
    null
}.get()
```


예제 10-3 ~ 6



IntegrationConfiguration

@Bean

```
fun etlFlow(@Value("\${input-directory:\${HOME}/in}") dir: File,
    c: BatchChannels,
    launcher: JobLauncher,
    job: Job): IntegrationFlow = IntegrationFlows
    .from(Files.inboundAdapter(dir).patternFilter("*.csv")) { cs ->
        cs.poller { p -> p.fixedRate(1000) }
    }.handle(File::class.java) { file, headers ->
        val absolutePath = file.absolutePath
        val params = JobParametersBuilder().addString("file", absolutePath)
            .toJobParameters()
```

MessageBuilder

```
        .withPayload(JobLaunchRequest(job, params))
        .setHeader(FileHeaders.ORIGINAL_FILE, absolutePath)
        .copyHeadersIfAbsent(headers)
        .build()
    }.handle(JobLaunchingGateway(launcher))
    .routeToRecipients { spec ->
        spec.recipient(c.invalid(), MessageSelector { notFinished(it) })
        .recipient(c.completed(), MessageSelector { finished(it) })
    }.get()
```

```
fun finished(msg: Message<*>): Boolean = JobExecution::class.java.cast(msg.payload)
```

FinishedFileFlowConfiguration

@Bean

```
fun finishedJobsFlow(channels: BatchChannels,  
    @Value("\${input-directory:\${HOME}/completed}") finished: File,  
    jdbcTemplate: JdbcTemplate): IntegrationFlow = IntegrationFlows  
    .from(channels.completed())  
    .handle(JobExecution::class.java) { _, headers ->  
        val ogFileName = headers[FileHeaders.ORIGINAL_FILE].toString()  
        val file = File(ogFileName)  
  
        mv(file, finished)  
  
        val contacts = jdbcTemplate.query("select * from CONTACT") { rs, _ ->  
            Contact(  
                rs.getString("full_name"),  
                rs.getString("email"),  
                rs.getBoolean("valid_email"),  
                rs.getLong("id"))  
        }  
        contacts.forEach(log::info)  
        null  
    }.get()
```

메시지 브로커, 브릿지, 경쟁적 컨슈머 패턴, 이벤트 소싱

메시지 브로커

브로커에 메시지가 유입되면 누군가 가져갈(소비) 때까지 저장

두가지 목적지 타입

- 점대점(point to point)

하나의 메시지를 단 하나의 컨슈머에게 전달

- 발행-구독(pub-sub)

유입된 하나의 메시지를 연결된 모든 구독자에게 전달

경쟁적 컨슈머 패턴: 점대점 메시징에서 브로커에 연결된 다수의 컨슈머가 유입된 메시지를 가져가서 처리하는 구조

이벤트 소싱: 도메인에서 발생하고 있는 모든 이벤트를 순서대로 로깅하고 이 로그를 새로운 시스템에 순서대로 주입해서 시스템의 상태를 원하는 시간으로 되돌릴 수 있는 패턴

스프링 클라우드 스트림

스프링 인티그레이션 기반으로, 서비스간 연동 모델에 메시지채널을 사용
메시징 기술을 쉽게 사용할 수 있도록 간단한 채널 정의 구현을 사용할 수 있다.
브로커 연결을 위해서 바인더 구현을 사용.

ex) 래빗엠큐를 지원하는 스프링 클라우드 스트림 바인더

```
org.springframework.cloud:spring-cloud-stream-binder-rabbit
```

스트림 프로듀서

바인딩: 다른 서비스와의 통신을 위한 `MessageChannel` 인스턴스로 통하는 논리 참조 제공
-> 스프링 클라우드 스트림에 의해 자동설정 됨

```
@Profile("producer")
interface ProducerChannels {

    companion object {
        const val DIRECT = "directGreetings"
        const val BROADCAST = "broadcastGreetings"
    }

    @Output(DIRECT)
    fun directGreetings(): MessageChannel

    @Output(BROADCAST)
    fun broadcastGreetings(): MessageChannel
}
```

`@Output(스트림 이름)` : 다른 서비스로 메시지를 보낼 채널 명시

application.yml

```
spring:
  cloud:
    stream:
      bindings:
        broadcastGreetings:
          destination: greetings-pub-sub
        directGreetings:
          destination: greetings-p2p

  rabbitmq:
    addresses: localhost
```

broadcastGreetings , directGreetings 는 메시지 채널 이름

greetings-pub-sub , greetings-p2p 는 프로듀서와 컨슈머가 만날 접점(랑데뷰 포인트)

스트림 프로듀서 - 직접 보내기

```
@SpringBootApplication
@EnableBinding(ProducerChannels::class)
class SimpleStreamProducer

@RestController
class GreetingProducer(channels: ProducerChannels) {
    val direct: MessageChannel by lazy { channels.directGreetings() }
    val broadcast: MessageChannel by lazy { channels.broadcastGreetings() }

    @RequestMapping("/hi/{name}")
    fun hi(@PathVariable name: String): ResponseEntity<String> {
        val message = "Hello, $name!"

        direct.send(MessageBuilder.withPayload("Direct: $message").build())
        broadcast.send(MessageBuilder.withPayload("Broadcast: $message").build())

        return ResponseEntity.ok(message)
    }
}
```


스트림 프로듀서 - 메시징 게이트웨이

```
@SpringBootApplication
@EnableBinding(ProducerChannels::class)
@IntegrationComponentScan
class StreamProducer

@Profile("producer")
@MessagingGateway
interface GreetingGateway {
    @Gateway(requestChannel = ProducerChannels.BROADCAST)
    fun broadcastGreet(msg: String)

    @Gateway(requestChannel = ProducerChannels.DIRECT)
    fun directGreet(msg: String)
}
```

스트림 컨슈머 - 메시지 채널

```
interface ConsumerChannels {  
  
    companion object {  
        const val DIRECTED = "directed"  
        const val BROADCASTS = "broadcasts"  
    }  
  
    @Input(DIRECTED)  
    fun directed(): SubscribableChannel  
  
    @Input(BROADCASTS)  
    fun broadcasts(): SubscribableChannel  
  
}
```

SubscribableChannel 은 MessageChannel 을 상속한 인터페이스.
구독/해지 매소드가 추가됨

application.yml

```
spring:
  cloud:
    stream:
      bindings:
        broadcasts:
          destination: greetings-pub-sub
      directed:
        destination: greetings-p2p
        group: greetings-p2p-group
        durableSubscription: true

rabbitmq:
  addresses: localhost
```

프로듀서에서 정의한 `destination` (랑데뷰 포인트)을 똑같이 지정해준다

`group` : 그룹에 속한 하나의 멤버만이 메시지를 받도록

`durableSubscription` : 메시지를 전달하지 못했을 때 컨슈머가 다시 접속할 때까지 저장했다가 전달하는 내구성 옵션

스트림 컨슈머 - 자바 DSL

```
fun incomingMessageFlow(incoming: SubscribableChannel, prefix: String): IntegrationFlow {
    return IntegrationFlows.from(incoming)
        .transform<ByteArray, String> { String(it) }
        .transform(String::class.java, String::toUpperCase)
        .handle(String::class.java) { greetings, _ ->
            log.info("greeting received in IntegrationFlow ($prefix): $greetings")
            null
        }.get()
}

@Bean
fun direct(channels: ConsumerChannels): IntegrationFlow =
    incomingMessageFlow(channels.directed(), "directed")

@Bean
fun broadcast(channels: ConsumerChannels): IntegrationFlow =
    incomingMessageFlow(channels.broadcasts(), "broadcasts")
```

메시지 콘텐츠 타입이 왜 ByteArray???

스트림 컨슈머 - 스트림 리스너

```
@StreamListener(ConsumerChannels.DIRECTED)
fun onNewDirectedGreetings(msg: String) {
    log.info("onNewDirectedGreetings: $msg")
}

@StreamListener(ConsumerChannels.BROADCASTS)
fun onNewBroadcastGreetings(msg: String) {
    log.info("onNewBroadcastGreetings: $msg")
}
```

```
brew services start rabbitmq-server
```

<http://localhost:15672/>