

Working with NoSQL Technologies / Spring Integration

Working with NoSQL Technologies

Spring Boot Features

Graceful shutdown is supported with all four embedded web servers (Jetty, Reactor Netty, Tomcat, and Undertow) and with both reactive and Servlet-based web applications. It occurs as part of closing the application context and is performed in the earliest phase of stopping SmartLifecycle beans.

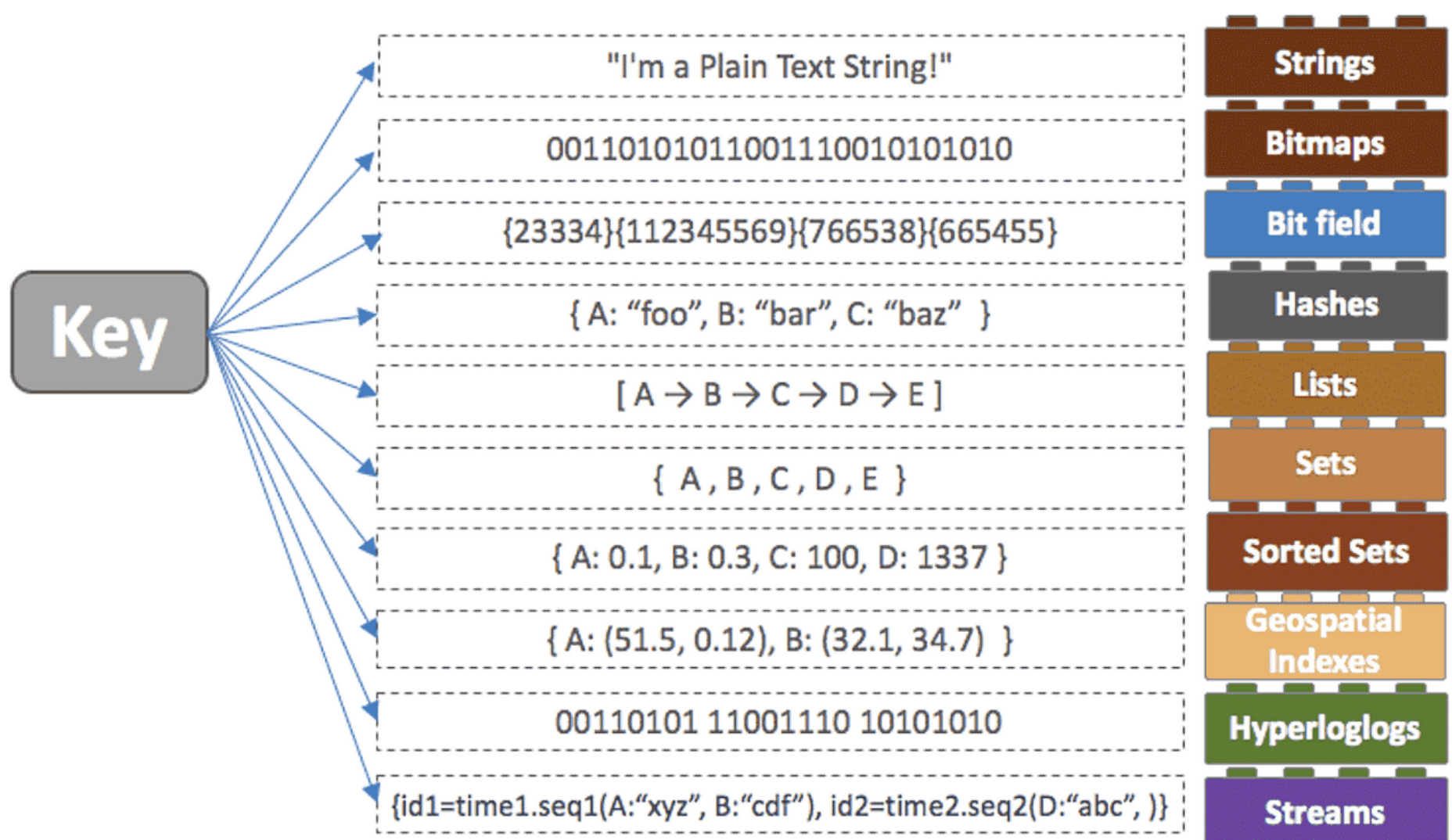
<https://docs.spring.io/spring-boot/docs/2.4.3/reference/html/spring-boot-features.html#boot-features-redis>

Redis

메모리 기반 저장 서비스로 Key-Value 구조를 가진다.

다양한 자료구조

다양한 자료 저장 형식을 지원한다.



- key는 문자열이며, 최대 512MB까지 가능하다.
- Key를 사용한 커맨드
 - `EXISTS <key>` : 해당 key 가 존재하는지 확인
 - `DEL <key1> <key2> ...` : key 삭제
 - `TYPE <key>` : key의 value 가 어떤 데이터타입을 사용하고 있는지 반환
 - `SCAN <cursor> [match pattern] [count]` : key의 목록을 커서단위로 얻어옴

Persistent

Redis는 데이터를 메모리에 저장한다. 즉, Shutdown 되면 모든 데이터가 날라갈 수 있다.

- 일반적인 Memcached처럼 사용할 수도 있지만, 휘발성을 해결하기 위해 영속성을 지원하기에 데이터를 보존 시킬 수도 있다.

AOF(Append Only File) 방식: 데이터 변경이 일어나는 커맨드를 AOF 파일에 기록하는 방식

- 입력/수정/삭제 (CUD) 명령이 실행될 때마다 .aof 파일에 기록된다.
- AOF를 계속 수행하면 기록되는 파일의 Size가 커져 기록이 중단될 수 있다.

RDB(Snapshot) 방식: Redis 인스턴스의 현재 메모리에 대한 dump (스냅샷) 를 생성하는 기능

- 특정한 간격으로 메모리에 있는 Redis 데이터를 바이너리 형태로 Disk에 기록
- AOF 방식보다 차지하는 크기가 적으며, 로딩하는 속도도 빠르다.

Spring Redis

Java에서는 Redis Client로 2가지가 존재한다.

- Jedis
- Lettuce

Spring Boot 2.0 이후로 `Lettuce` 가 기본 클라이언트가 되었다. 고로, `spring-boot-starter-data-redis` 를 추가해서 사용하면 기본적으로 `lettuce` 를 사용할 수 있다.

- Jedis는 Lettuce에 비해 성능이 좋지 않다고 하다. [여기](#)를 참조하자.

만약 reactive로 구현하고 싶다면 `spring-boot-starter-data-redis-reactive` 를 사용하자.

Cache 기능 지원

- **@EnableCaching:** Cache annotation public method의 모든 스프링 빈을 스캔. 해당 메서드들을 intercept 하여 캐싱 기능이 추가된 프록시를 생성.
- **@Cacheable:** 캐시 생성 수행.
- **@CacheEvict:** 캐시 삭제 수행.
- **@CachePut:** 캐시 업데이트 수행(메서드 실행에는 영향을 끼치지 않으며).
- **@Caching:** 메서드에 적용할 캐시 작업 그룹 정의.
- **@CacheConfig:** 클래스 레벨에서 캐시 관련 설정을 공유.

```
@GetMapping()
@Cacheable("user")
public String get(@RequestParam("id") String id) {
    logger.info("get user - userId:{},", id);
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
    return id;
}
```

```
127.0.0.1:6379> keys *  
1) "user::id"
```

Spring Integration

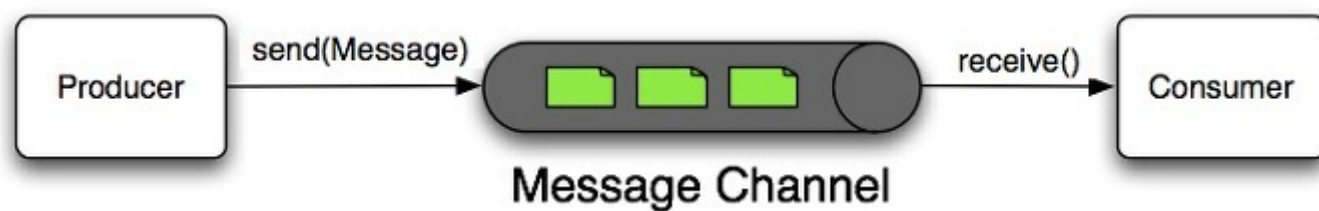
Spring application 내 메시징 또는 HTTP, TCP 같은 다른 전송 프로토콜에 대한 추상 계층을 제공한다.

→ **외부 시스템과의 통합**을 쉽게 할 수 있도록 도와준다.

→ One of the most important goals of Spring Integration is to provide a simple model for **building maintainable and testable enterprise integration solutions**.

결국, 여러 통신을 메시징 기반으로 통합해 구현했고, 여기에 패턴화도 구성되어 있는 것.

Spring Integration



통합 패턴 사용 → 메시지를 전달하는 컴포넌트들을 조립하여 파이프라인을 구성한다.

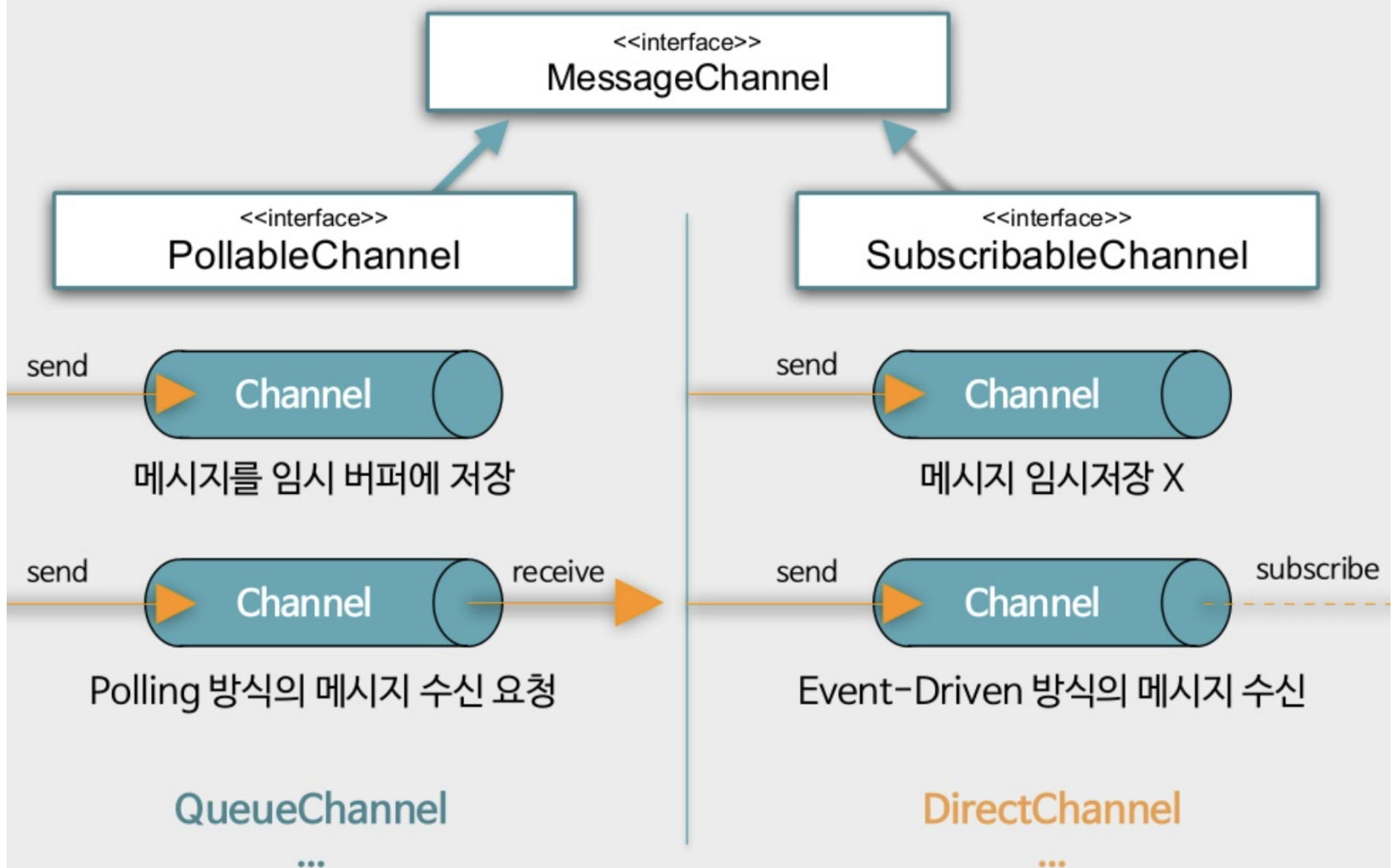
- 구성요소: Message, Channel, Endpoint

Message: 목적지로 보내는 데이터 (Wrapper)

- Header와 Payload로 구성
 - Header: 메시지의 메타 데이터
 - Payload: 전송하고자 하는 데이터

Channel: 한 요소로부터 다른 요소로 메시지를 전달

Message Channel 분류



Message Channel 분류



Endpoint: 메시지 채널에서 메세지를 다양한 방법을 통해 처리

- Filter: 조건에 맞는 메시지가 플로우를 통과하게 해줌
- Transformer: 메시지 값을 변경하거나 메시지 페이로드의 타입을 다른 타입으로 변환
- Router: 여러 채널 중 하나로 메시지를 전달하며 대개 메시지 헤더를 기반으로 함
- Splitter: 들어오는 메시지를 두 개 이상의 메시지로 분할하며, 분할된 각 메시지는 다른 채널로 전송
- Aggregator: 분배기와 상반된 것으로 별개의 채널로부터 전달되는 다수의 메시지를 하나의 메시지로 결합함
- Service Activator: 메시지를 처리하도록 자바 메서드에 메시지를 넘겨준 후 메서드의 반환값을 출력 채널로 전송 → 메시지가 입력 채널에 도착했을 때, 특정 메서드 호출
- Channel Adapter: 외부 시스템에 채널을 연결함. 외부 시스템으로부터 입력을 받거나 쓸 수 있음
- Gateway: 인터페이스를 통해 통합 플로우로 데이터를 전달

예제

```
public Properties mailProperties() {
    Properties properties = new Properties();
    properties.setProperty("mail.imap.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
    properties.setProperty("mail.imap.socketFactory.fallback", "false");
    properties.setProperty("mail.store.protocol", "imaps");
    properties.setProperty("mail.debug", "true");
    properties.setProperty("mail.imap.ssl", "true");
    return properties;
}

@Bean
public ImapIdleChannelAdapter imapIdleChannelAdapter(MessageChannel emailInChannel) {
    // Define mail receiver
    ImapMailReceiver imapMailReceiver = new ImapMailReceiver(createImapUrl());
    imapMailReceiver.setJavaMailProperties(mailProperties());
    imapMailReceiver.setShouldMarkMessagesAsRead(false);
    imapMailReceiver.setShouldDeleteMessages(false);
    imapMailReceiver.afterPropertiesSet();

    // Define channel adapter
    ImapIdleChannelAdapter imapIdleChannelAdapter = new ImapIdleChannelAdapter(imapMailReceiver);
    imapIdleChannelAdapter.setAutoStartup(true);
    imapIdleChannelAdapter.setOutputChannel(emailInChannel);
    imapIdleChannelAdapter.afterPropertiesSet();
    return imapIdleChannelAdapter;
}

@ServiceActivator(inputChannel = "emailInChannel")
public void handleMessage(MimeMessage message) throws IOException, MessagingException {
    System.out.println("### Message Received!!!");
    System.out.println("### " + message.getSubject());
    System.out.println("### " + message.getFrom()[0]);
    System.out.println("### " + message.getContent().toString());
}

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

private String createImapUrl() {
    return "imaps://<user-email>:<password>@imap.gmail.com:993/inbox";
}
```

Reference

- <https://sjh836.tistory.com/178>
- <https://ckddn9496.tistory.com/107>
- <https://sup2is.github.io/2020/08/12/what-is-spring-integration.html>

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