범용 메시징 통합 처리 설계 및 구현

손호성, 유헌창
고려대학교 컴퓨터정보통신대학원
e-mail : {pennori, yuhc}@korea.ac.kr

Design and Implementation of Universal Messaging Integration Processing

Ho-Seong Son, Heonchang Yu
Graduate School of Computer & Information Technology, Korea University

요 약

IT 콘텐츠 소비 주체 유입 규모는 지속적으로 증가할 뿐만 아니라, 다루는 데이터의 크기가 커지고, 트래픽 또한 기하급수적으로 늘어난다. 비즈니스 분류도 세분화되고 콘텐츠가 소비되는 속도도 점점 가속화되고 있다. 현대의 애플리케이션은 레거시나 신규를 가릴 것 없이 지속적으로 사용자의 기호에 맞는 새로운 서비스를 제공하기 위해서 매우 기민한 개발 프로세스를 가질 필요가 있다. 따라서 이 같은 문제를 해결하기 위해 본 논문에서는 개발자들이 자주 쓰는 필수적이고 반복적인 기능과 개념들을 한데 모아 구축한 인프라 스트럭처와 이를 기반으로 서비스 규모를 최소화하고 관계를 통해 확장하는 아키텍처를 제안하였다. 제안하는 아키텍처는 비즈니스를 쉽고 빠르게 최소 단위 애플리케이션으로 구현하도록 지원함으로 인해, 개발에 소요되는 비용을 줄일 수 있다.

1. 서론

IT 콘텐츠 소비 주체 유입 규모는 꾸준히 증가중이다. 소비의 파이가 커지면서 기호가 다양해짐과 동시에 이들 IT 콘텐츠 소비 주체인 고객의 눈높이는 날로 높아져가고 있다. 다루는 데이터의 크기가 커지고, 트래픽 또한 기하급수적으로 늘어난다. 대체로 이런 상황에서 해결책으로 지목되는 것이 작은 서비스와 분산처리다. 그런데 일반에 공개된 내용은 턱없이 적어서 성능이나 효용성에 대한 검증이 사실상 어렵다. 그러다보니 이들 애플리케이션 개발자들은 늘 0 부터 시작하는 패턴을 보이고 있다. 예를 들면, 목적지로 이동할 때 반드시 자동차를 타고 이동해야 한다고 할 때, 늘 자동차를 새로 만들거나 조립할 파츠를 잔뜩 메고 다니다 조립해서 가고 있는 것이다. 목적지에 도착하기 전에 진이 빠져버릴 수도 있다. 이런 비효율적인 일들이 실무에서 심심치 않게 발생하고 있다. 이후 관련 연구에 기술될 스프링 부트처럼 필수적이고 반복적인 기능과 개념들을 한데 모아 인프라 스트럭쳐를 구축함으로써 개발에 들어가는 수고를 줄이고자 한다.

2. 범용 메시징 통합처리의 요구사항

2.1 스프링 부트 (Spring boot)

스프링 부트는 스프링 프레임워크 서드파티 프로젝트 중 하나이다. 스프링 프레임워크는 서비스 코드를 구현하기 위한 설정이 꽤 복잡하다. 자바 애노테이션이 아닌 XML 로 기술할 경우 디버깅 애로사항도 배가된다. 스프링 부트는 스프링 프레임워크 기반 애플리케이션을 개발할 때 환경설정 면에서 극도의 편의를 제공하게 하는 구조로 되어있다. 필수적이지만 반복적인 수 많은 설정을 자동화해서 스프링 부트를 사용하지 않을 때보다 훨씬 쉽고 간단하게 사용할 수 있게 한다. 노력이 절약되는 것이다. 기능 면에서 눈에 띄는 특징은 크게 3 가지가 있다.

(1) xml 설정을 필요로 하지 않는다.
(2) Tomcat, Jetty, Undertow 등의 서블릿 컨테이너가 내장되어 외부의 도움없이 서비스를 실행할 수 있다. (기본값으로 설정되어 있으며 해제하고 기존 방식대로 실행도 가능하다.)
(3) Maven 을 사용하는 경우 사용량이 많은 라이브러리를 집약한 스타터(Starter) POM 파일로 설정에 편의를 가져올 수 있다.

2.2 스프링 인티그레이션 (Spring Integration)

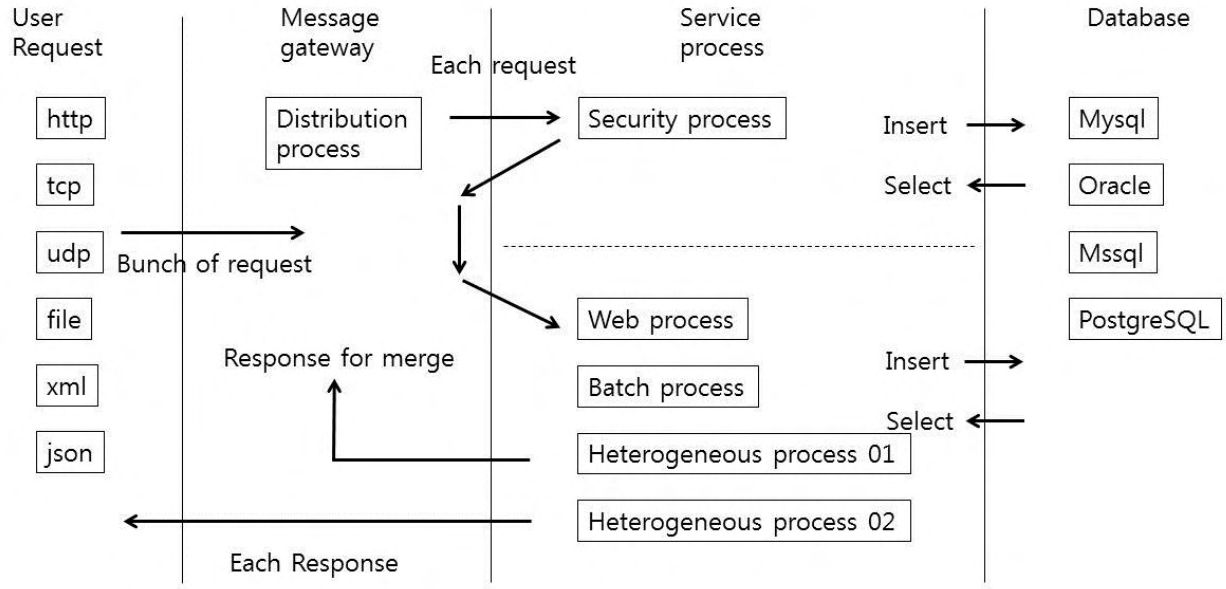
스프링 인티그레이션은 스프링 프레임워크 서드파티 프로젝트 중 하나이다. 스프링 인티그레이션은 스프링 프레임워크 기반 애플리케이션을 개발할 때 메시지 기반 통신에 편의를 제공한다. 사실 스프링 인티그레이션에서 지원하는 개념과 기능은 기존에 모두 존재하는 것들이다. 다만 스프링 인티그레이션은 기존에 존재하는 동기/비동기, 단방향/양방향, 푸시/풀

링 등의 메시지 기반의 통신 행위 구현을 보다 수월하게 하는 것이다. 스프링 인티그레이션 역시 기본적으로 스프링 프레임워크에서 지향하는 객체지향적 개발원칙을 지키게끔 유도한다. 대략적인 내용은 다음과 같다.

(1) 관점지향 프로그래밍을 강제한다. 서비스와 공통로직은 분리되고 서로 관여하지 않는다.
(2) 테스트와 유지보수성 향상을 위해 개방폐쇄(OCP) 원칙을 강제한다.
(3) 확장이 필요한 곳은 추상화를 권장한다. 기능 측면에서 선언적 어댑터 API 를 제공한다. 외부 시스템과의 연계가 수월해지고 재사용성이 상승하게 된다.

2.3 자바 쓰레드 어피니티(Java Thread Affinity)

자바 쓰레드 어피니티는 OpenHFT 프로젝트 중 하나이다. 자바를 기반으로 한 고빈도매매 (고성능 컴퓨터를 이용한 초단타 매입) 를 지원하는 라이브러리다. 자바에서 기본적으로 제공하는 API 를 사용하는 경우 특정 CPU 로의 부하는 피할 수가 없다. 자바 쓰레드 어피니티는 OS 에서 제공하는 CPU 정보를 이용해 쓰레드를 논리 CPU 혹은 전체 코어에 할당하는 기능을 수행한다.



(그림 1) 시스템 구성도

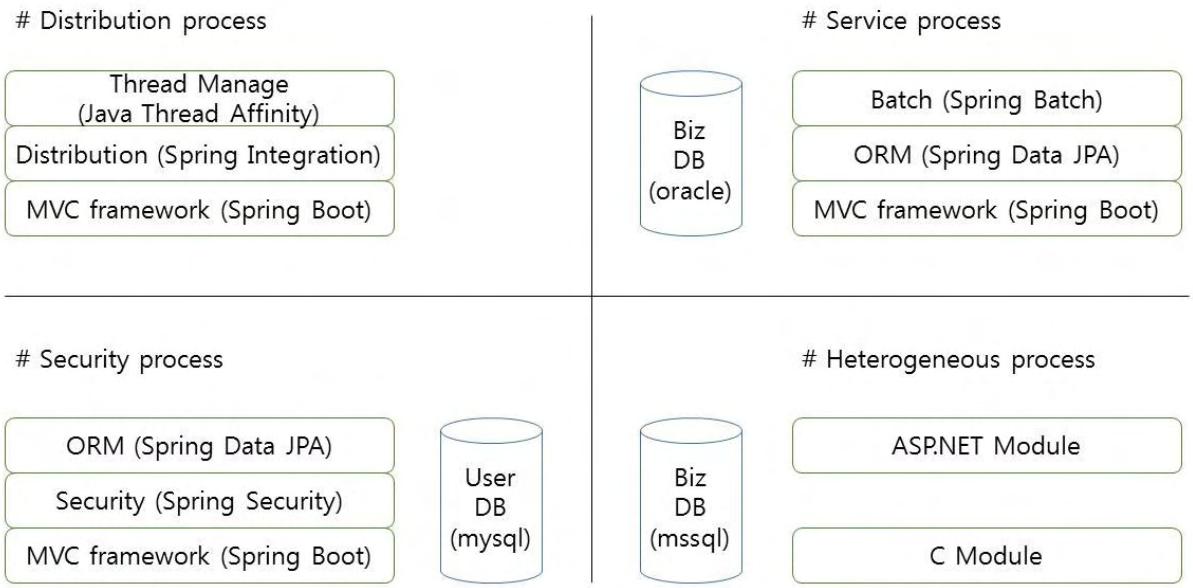
3. 범용 메시징 통합처리 설계

3.1 플로우 설계

(그림 1)은 제안하는 시스템 구성도이다. 각 계층은 역할에 따라 4 단계로 분류되었다. 요청은 다양한 경로와 방식이 대량으로 접수될 수 있음을 상정하였다. 프로토콜은 표준 형식을 지향한다. 메시지 게이트웨이는 요청을 취합/분류한다. 쓰레드 처리 단위에 따라 요청을 대기시키거나 비동기 작업으로 위임하기도 한다. 보안 프로세스는 접근할 자원에 따라 인증을 거쳐 권한을 체크/부여한다. 보안 프로세스를 거친 후 실제 요청에 대한 서비스 프로세스는 기능과 도메인, 프로토콜 등에 따라 최적화된 단위의 소규모 모듈이 대응하게 된다. 기본적으로 각 서비스 프로세스에 해당하는 모듈들은 독립적으로 동작한다. 보안 부분만 유일하게 광범위한 의존관계를 가지게 된다. 주어진 역할에 따라 요청에 대응하며 응답 또한 취합/분할/개별 응답이 가능하다. 데이터베이스는 JPA, querydsl 을 ORM 으로 사용하여 MySQL, Oracle, Ms-SQL, PostgreSQL 등 어떤 RDBMS 로도 구성 가능하다.

3.2 개별 구성모듈 예시

(그림 2)는 구성 가능한 개별 모듈의 예시다.
기능과 도메인, 프로토콜에 따라 소규모 단위로 분할되어 있다. 다른 시스템과 전혀 관계없이 추가가 가능하다. 다른 기종간의 통신도 고려되어 있다(전송시 실제 데이터는 직렬화된 바이트 배열로 보내진다. 다만 가급적 데이터 내용 중 언어에 따라 해석이 다른 short 같은 자료형은 취급하지 않는다). 다만 통합 관제는 선택사항이다. 같은 계통의 기술 스택이라면 조금 더 쉽게 통합관제가 가능하다.



(그림 2) 개별 모듈 구성도

4. 범용 메시징 통합처리 구현

4.1 구현 환경

시스템 구현을 진행하기 위해 다음과 같은 환경을 구성하였다. 먼저 주 개발 언어는 Java 8 (플랫폼에 독립적인 Java 언어의 특성상 OS 가 Windows 건 Unix 건 영향을 받지 않는다.), IDE 는 이클립스 Neon 버전을 베이스로 하는 STS 3.8.0 버전 (STS 는 이클립스 바탕에 스프링 프레임워크 사용에 편의를 더한 IDE) 을 이용하였다. 연동할 라이브러리로 스프링 부트 (스프링 프레임워크 바탕), 스프링 인티그레이션, 자바 쓰레드 어피니티, 스프링 데이터 JPA, H2/MySQL DBMS 를 연동하여 구현하였다. (H2 는 내장형 DBMS 로 애플리케이션과 동일한 생명주기를 갖도록 구성하였다. 초기 설정시 DBMS 설정에 관계없이 편의를 도모하기 위한 구성이다. 운영환경에서는 권장하지 않는다.)

4.2 적용된 다양한 프레임워크

시스템의 효율적인 동작과 자원관리, 소스 코드의 품질재고를 위해서 스프링 프레임워크 외에도 Git (형상관리), Junit (단위 테스트), DBUnit (퍼시스턴스 계층 단위 테스트) 등의 다양한 프레임워크를 이용하였다.

(1) Git – 형상관리

다수의 인원이 팀 프로젝트를 진행하며 야기될 수 있는 소스코드 관리 상의 문제 (충돌, 현행화 미흡) 를 해결하게 하는 형상관리도구이다. 비슷한 역할을 하는 것으로 CVS, SVN 등이 있으며 이들보다 후 세대로 등장했고 전 세대의 단점을 상당수 보완하였다.

(2) Junit – 단위테스트, 아키텍처의 구현 유효성 검증

Java 개발에서 단위 테스트 개념을 이야기할 때 빠지지 않고 등장하는 단위 테스트 도구이다. 행위 기반의 테스트를 유도하며 수행시간의 관리도 가능하게 한다. 스프링 프레임워크와의 연동을 통해 스프링 빈의 테스트도 지원한다.

(3) DBUnit – DB 연동 단위테스트 지원

DB 를 이용한 테스트가 필요할 때, 테스트 수행 전에 필요한 데이터를 미리 저장해놓고 테스트 종료 후 이를 삭제하는/초기화하는 작업을 더 편리하게 수행하게 하는 테스트 프레임워크이다. 가장 큰 장점으로 테스트를 위해 실제 소스에 불필요한 코드를 추가하는 것을 방지해주는 점을 꼽을 수 있다. 스프링 테스트 디비유닛과 연동을 통해 스프링 빈으로 등록/주입되는 데이터 소스에 대한 테스트도 지원한다.

5. 범용 메시징 통합처리 구현 도입시의 효과 분석

현재 본 논문에서 제안하는 개념과 구성은 실무에서 동일하게 사용되는 사례가 없다. 그래서 사례 연구는 가상으로 설정한 상황에서, 제안하는 처리 프로세스를 사용하는 경우와 아닌 경우의 편의를 비교하였다.

5.1 구현 상의 편의 분석

(그림 3)은 TCP 서버를 구현하는 작업을 놓고 각각 방법을 달리 해서 구현한 코드 예시이다. 상단이 미적용, 하단이 적용된 상태이다. 미적용시의 코드는 일단 가독성이 떨어진다. 성격이 다른 여러 역할이 강하게 결합되었기 때문이다. 서버의 생명주기도 개발자가 세세하게 정의하게 되어 있다.

Disabled

```
Try {
  ServerSocket serverSocket = new
ServerSocket(port);
  Socket socket = serverSocket.accept();

  OutputStream out =
socket.getOutputStream();
  InputStream in = socket.getInputStream();
  PrintWriter pw = new PrintWriter(new
OutputStreamWriter(out));
  BufferedReader br = new
BufferedReader(new InputStreamReader(in));
String request = null;
  while ((request = br.readLine()) != null) {
    ..do something..
    pw.flush();
  }

  pw.close();
  br.close();
  socket.close();
} catch {
..Do something..
}
```

Enabled

```
@Bean
public TcpReceivingChannelAdapter
plainAdapter() {
  TcpReceivingChannelAdapter adapter =
new TcpReceivingChannelAdapter();

adapter.setConnectionFactory(plainServerFa
ctory());

adapter.setOutputChannel(inputWithPlain());

  return adapter;
}
@Bean
public AbstractConnectionFactory
plainServerFactory() {
  int port = Integer.parseInt(inboundPort);
  TcpNioServerConnectionFactory factory =
new TcpNioServerConnectionFactory(port);

  return factory;
}
@Bean
public Executor taskSchedulerWithPlain() {
  ThreadPoolTaskScheduler scheduler = new
ThreadPoolTaskScheduler();
  scheduler.setPoolSize(PROCESS_SIZE);

scheduler.setThreadFactory(plainAffinityThreadFa
ctory());

scheduler.setWaitForTasksToCompleteOnShutdo
wn(true);

  return scheduler;
}
```

(그림 3) 미적용 vs 적용시 코드 예시

개발자의 작업 이해도에 따라 코드의 품질이 좌우된다. 확장을 시도할 시 재사용이 불가하거나 코드 중복이 발생할 우려가 있다. 적용시 코드는 하나의 서버가 게이트웨이, 커넥션관리, 작업스케줄링 등 모듈 단위로 분리되어 있다. 각 모듈은 스프링 빈으로 선언하여 스프링 컨테이너가 생명주기를 관리한다. 이 환경에선 사용된 모든 상수와 변수의 프로퍼티화가 가능해지는 장점이 있다. 코드 품질 또한 일정하게 유지된다.

6. 결론

이 논문에서는 스프링 프레임워크와 서드파티에서 제공하는 자동화된 구성을 이용해서 빠르게 웹 애플리케이션을 만들어 보았다. 무에서부터 시작하는 방식과 비교해 보니 편의는 물론 코드의 복잡성과 확장성 측면에서도 차이가 나타났다.

향후 연구에서는 구현된 코드의 안정성을 검증하는 테스트 방법에 대하여 연구할 것이다.

참고문헌

[1] Radu Bucea-Manea-Ţoniş, Rocsana Tonis (Bucea-Manea), “HOW TO DESIGN A WEB SURVEY USING SPRING BOOT WITH MYSQL: A ROMANIAN NETWORK CASE STUDY, Annals of Spiru Haret University, Economic Series, Vol.17, Issue2, 63-72, 2017.

[2] 이한성, 조보현, 김희중, 신지영 ” Spring Boot 를 활용한 웹 기반의 종합 프로젝트 관리 시스템”, 한국정보과학회 2016 년 동계학술대회 논문집, 116-118, 2016.

[3] http://spring.io

[4] http://junit.org

[5] http://dbunit.sourceforge.net

[6] https://github.com/OpenHFT/Java-Thread-Affinity