Hanbit eBook

Realtime 92

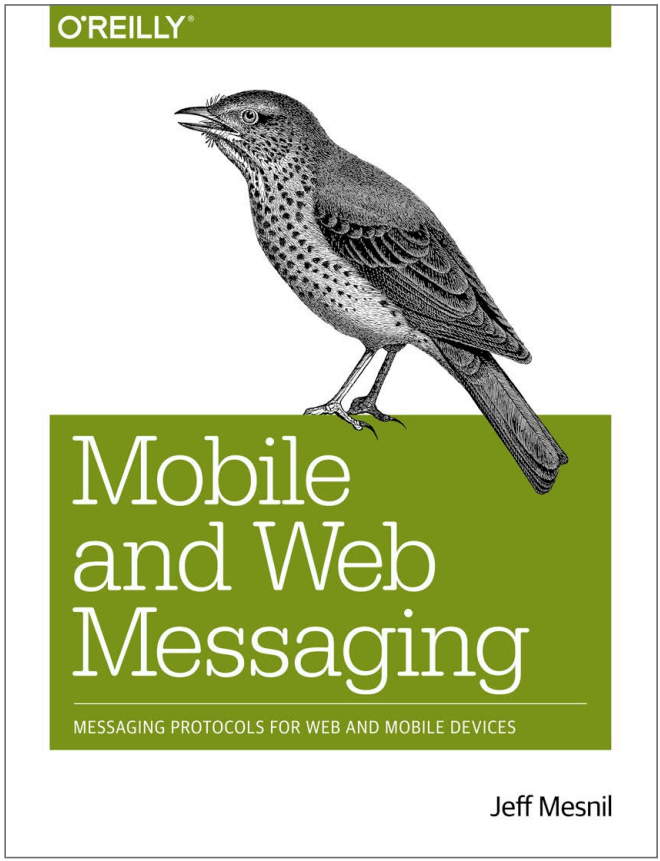
모바일/웹 메시징

STOMP와 MQTT로 개발하는 IoT 모바일/웹 애플리케이션

Mobile and Web Messaging

제프 메스닐 지음 / 조건희 옮김

O'REILLY® 한빛미디어 Hanbit Media, Inc.



이 도서는 O'REILLY의
Mobile and Web Messaging의
번역서입니다.

모바일/웹 메시징: STOMP와 MQTT로 개발하는 IoT 모바일/웹 애플리케이션

초판발행 2015년 2월 27일

지은이 제프 메스닐 / **옮긴이** 조건희 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-741-5 15000 / **정가** 14,000원

총괄 배용석 / **책임편집** 김창수 / **기획·편집** 김상민
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 최송실
영업 김형진, 김진불, 조유미 / **마케팅** 박상용

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

저자 소개

저자_ **제프 메스닐**

제프 메스닐Jeff Mesnil은 여러 오픈 소스 미들웨어 프로젝트에 참여하고 있는 소프트웨어 개발자다. 현재는 레드 햇Red Hat에서 시니어 엔지니어로 일하고 있다.

역자 소개

역자_ **조건희**

대한민국의 평범한 개발자로, 다양한 문제를 해결하는 것에 즐거움을 느끼며 현업에 종사하고 있다. RoR로 서비스를 만들기 시작하여, 현재는 스프링, Backbone, RequireJS, 파이썬 등을 통한 솔루션 개발에 한창이다. 한 줄 한 줄 작성한 코드가 사용자에게 가치를 전달한다는 점에서 하는 일에 매력을 느끼며, 개발자의 가치와 생산성을 높이는 일에도 관심이 많다.

저자 서문

작은 스마트 디바이스 사용이 점차 증가하는 추세다. 스마트폰과 태블릿은 인터넷에 연결해 사용하고, 자동화 디바이스, 가정용 디바이스, 자동차 시스템 등이 이에 합세하고 있다. 비록 이 디바이스들이 갈수록 강력해지고는 있지만, 여전히 데스크톱 디바이스에 비해서는 제한된 배터리 수명과 메모리, 간헐적인 네트워크 연결 불가 문제를 포함한 많은 제약이 있다. 이들은 다른 디바이스들과 통신하면서도 배터리와 메모리 사용량을 보존하기 위해 효율적인 프로토콜이 있어야 한다.

메시징 기술은 우리에게 이러한 제약을 극복할 수 있게 해준다. 메시징에서 제공되는 프로토콜은 반드시 필요한 리소스(네트워크 대역폭과 메모리 사용)만 사용하며, 네트워크 연결이 끊어지는 경우에도 데이터가 효과적으로 전달될 수 있게 보장하기 때문이다.

점점 많은 애플리케이션이 웹 브라우저나 웹 기술을 이용해 동작하고 있다. 메시징 개념은 이러한 애플리케이션에도 같이 적용되며, 이들이 좀 더 효율적이고 빠르게 응답할 수 있게 도와준다.

메시징은 엔터프라이즈 소프트웨어 도메인 분야에서 널리 알려진 개념이지만, 모바일과 웹 컴퓨팅을 포함한 다른 도메인에서는 이제 막 시작하는 단계에 불과하다. 이 도메인에서 소프트웨어를 만드는 개발자들에게는 메시징 기술들을 잘 다루기 위한 좋은 안내서가 필요하며, 이 책은 효과적으로 메시징을 사용할 수 있게 도와줄 것이다.

이 책은 메시징 프로토콜들을 소개하여 소프트웨어 개발자들이 모바일과 웹 애플리케이션에 사용할 수 있게 한다.

역자 서문

모바일과 웹 간의 통신이 이루어지는 개발을 처음 경험한 때가 2011년이었다. 4년이 지난 지금 이 책의 주제는 내가 느끼는 것처럼 다른 사람들에게도 그리 신선한 주제가 아닐 수도 있다. 하지만 MQTT, STOMP를 비롯한 XMPP, AMQP 등 경량의 메시징 프로토콜들은 여전히 많은 곳에서 활용되고 있으며, 점점 더 늘어가는 추세다. 스마트폰이 아닌 다른 가상 현실 기기들이 우리의 몸과 주변에서 함께할 날도 그리 멀지 않았다. MQTT와 STOMP를 통해 경량 메시징 프로토콜의 개념들을 소개하는 이 책은 누구나 쉽게 읽을 수 있는 좋은 입문서다. 더불어 웹소켓 등을 통해 웹과 iOS가 통신하는 하나의 완전한 예제를 함께 다룬다. 아직 경량의 메시징 프로토콜들을 접해본 적이 없거나, 기본적인 것들을 다시 한 번 짚어보고 싶은 분들에게 이 책은 도움이 되리라 확신한다.

일러두기

이 책에서 다루는 것

이 책은 메시징 프로토콜을 다루고 있으며, 메시징을 활용해서 더욱 응답이 빠르고 오류가 발생해도 프로그램을 수행할 수 있는 애플리케이션(모바일 디바이스와 웹 브라우저에서 동작하는)을 만드는 방법을 설명한다.

메시징 프로토콜은 새로울 것이 없다. 이들은 엔터프라이즈 소프트웨어 분야에서 오랫동안 성공적으로 쓰여져 왔고, 서로 다른 서비스와 플랫폼이 통신할 수 있게 해주는 하나의 요소였다. 메시징 프로토콜의 설계는 이들이 모바일 디바이스 및 웹에서 동작하는 애플리케이션을 만드는 데도 사용될 수 있도록 해준다.

최근 들어서는 HTTP가 전송 프로토콜의 대세로 부상했고, 클라이언트(웹 브라우저에서부터 데스크톱, 모바일 애플리케이션, 백엔드 서비스 등)와 웹 서버 간 통신에 널리 쓰이고 있다. 이는 거의 모든 전매 특허와 비표준 프로토콜들을 대체했으며, 애플리케이션이 다른 원격 지점과 통신하고자 할 때 많이 선택하고 있다.

메시징 프로토콜은 HTTP를 보완해 주지만, 이 책은 모바일이나 웹 애플리케이션을 만드는 데 있어 메시징 프로토콜이 HTTP보다 더 적합한 경우에 중점을 두고 있다.

이 책에서 다루지 않는 것

메시징은 소프트웨어 개발에 있어 많은 것들을 가리킬 수 있는 광범위한 용어다. 이 책에서는 애플리케이션 메시징 프로토콜을 의미하는 용어로만 사용한다.

이 책은 애플의 메신저나 왓츠앱WhatsApp과 같은 메시징 애플리케이션들을 다루지는 않는다. 물론 이 애플리케이션들은 메시징 프로토콜을 기반으로 만들어지며 이 책은 이들을 만들기 위한 입문서로 괜찮을 것이다. 하지만 메시징 애플리케이션들

에서 기대되는 다른 많은 특징들을 다루지는 않는다.

또한, 이 책은 프로그래밍 언어나 오브젝티브-C에서 객체의 메서드를 작동시키거나 얼랭Erlang에서 프로세스들 간의 통신을 위해 사용되는 프래임워크 메시징을 다루지 않는다.

지금부터 메시징이라는 용어는 애플리케이션 메시징 프로토콜에서의 메시징을 가리키는 것으로 사용한다.

메시징은 간단하다

메시징 프로토콜의 개념은 간단하다.

- 애플리케이션은 중개자broker를 통해 목적지destination로 메시지를 공급produce한다.
- 애플리케이션은 메시지를 받기consume 위해 이 같은 목적지를 구독subscribe한다.

이 2개의 문장에서는 메시징을 설명하는 5개의 개념[01]이 소개되고 있다.

메시징 프로토콜은 간단한 아이디어다. 이를 사용하는 데 있어 가장 복잡한 부분은 여러분의 애플리케이션에 공급자와 소비자가 어떻게 메시지를 주고받을지 적합한 모델을 설정하는 일이다. 이 책은 가장 흔하게 사용되는 점대점, 게시/구독 모델을 소개한다.

---

01 역자주_ 윗첨자가 병기된 단어들 즉, 중개자, 목적지, 공급, 받기, 구독을 가리킨다.

엔터프라이즈 메시징은 간단하지 않다

회사들이 인수되거나 합병될 때는 각자가 가지고 있던 기존 시스템들끼리 서로 통신할 수 있는 방법이 필요하다. 메시징은 이러한 통합을 가능한 한 잡음 없이 완료될 수 있게 해주는 하나의 접근법이다. 시스템들은 메시지에 담겨 전달되는 데이터 표현data representation과 목적지destination 혹은 서로 다른 시스템들이 공유하는 관심 토픽에 동의해야 한다.

메시징 프로토콜이 엔터프라이즈 소프트웨어에서 사용될 때는 엔터프라이즈 요구사항(고가용성high availability, 장애 복구failover, 로드 밸런싱load balancing 등)을 만족시키기 위해 그 복잡도가 증가된다.

게다가 통합된 애플리케이션을 회사 내에서 사용하기 위해 메시징 시스템에 동의해야 한다. 자바 세상에서는 메시징을 다루는 명세를 자바 메시징 서비스Java Messaging Service[02]라고 부른다. 이는 자바 애플리케이션이 메시지를 주고받기 위해 사용할 수 있는 몇 개의 인터페이스 집합을 정의하고 있다.

그러나 JMS는 와이어wire[03]를 통해 바이트들이 어떻게 보내져야 하는지에 관한 어떤 프로토콜도 정의하지 않는다. 대신에 구현의 세부 사항을 API를 구현하는 JMS 중개자에게 넘긴다. 이는 JMS 구현이 상호 운용 가능하지 않다는 것을 의미한다. 중개자로 메시지를 보내기 위해서는 중개자의 클라이언트 구현을 사용해야만 한다. 애플리케이션들이 서로 다른 JMS 중개자를 사용했다면 상호 운용 이슈가 발생할 수밖에 없고 JMS 중개자 사이에 이들을 연결해주는 중간 단계가 있어야 한다.

---

02 http://bit.ly/jms-spec

03 역자주_ 와이어는 메시지를 주고받을 수 있는 통신 수단을 가리킨다.

이러한 상호 운용의 문제는 중간 단계 추가 및 각 중개자의 가용성을 보장하는 등의 복잡한 과정을 거치게 한다.

우리는 일정 기간 동안 엔터프라이즈의 기능과 상호 운용성을 다루는 Advanced Message Queuing Protocol[04]과 같은 엔터프라이즈 메시징 프로토콜들의 모습을 보아왔다. 이들은 구현하기도 어렵고 상호 운용성에서도 하위 호환성이 보장되지 않고 서로 다른 구현들은 전체 명세를 만족시키지 못하는 등 지속적으로 문제가 발생했다.

다시, 모바일 메시징은 단순하다

엔터프라이즈 소프트웨어에서 사용되는 메시징 프로토콜은 복잡도 증가로 인해 구현이 어렵고, 모바일 디바이스처럼 제약이 많은 환경에 적합하지 못하다.

모바일 디바이스들이 필요로 하는 메시징 프로토콜들은 좀 더 제한된 목적을 가진, 배터리를 덜 소모시키고 네트워크가 항상 연결될 필요도 없는, 사용하기에 단순하고 효율적인 것들이다.

비록 이 간단한 프로토콜들이 엔터프라이즈 수준의 많은 기능을 제공하지는 못하겠지만 모바일과 웹 플랫폼에는 아주 잘 들어맞는다.

이 책에서는 두 가지를 언급한다. 바로 STOMP와 MQTT다.

텍스트 메시지를 특정 시스템(운영 체제, 가상화 디바이스, 웹 브라우저)에서 다른 곳으로 보내고 싶은 경우 STOMP는 단순함 측면에서 큰 강점을 가진다. 이 프로토콜은 너

---

04 http://www.amqp.org/

무 단순해서 텔넷 같은 네트워크 클라이언트조차 STOMP 클라이언트가 될 수 있다.

MQTT는 적은 전력과 제약이 많은 메모리 자원을 가진 작은 디바이스들에서 데이터를 동보하기 위해 만들어졌다. 배터리 수명과 메모리를 잘 보존하기 때문에 모바일 디바이스에 적합하다.

이러한 프로토콜들은 이해하기도 구현하기도 쉽다. 메시징 중개자들은 종종 이 둘 모두를 제공한다. 예를 들어 데스크톱 애플리케이션은 STOMP를 사용해서 메시지를 보내고 다른 모바일 애플리케이션들은 같은 메시지를 MQTT를 사용해서 소비할 수 있다. 애플리케이션들은 자신들의 필요를 가장 잘 충족할 수 있는 메시징 프로토콜을 자유롭게 사용할 수 있으며 상호 운용성을 위해 중개자의 구현에 의존한다.

모바일 디바이스의 출현으로 이 간단한 프로토콜들을 사용해서 좀 더 반응적이고 효율적인 애플리케이션을 만들 수 있게 되었다. 이 프로토콜들은 모바일과 웹 플랫폼 모두에서 이용 가능하며 어느 것을 선택하든 상관 없다.

이와 더불어 메시징 프로토콜들은 레거시 시스템과도 함께 통합되어 사용될 수 있다. 만약 메시징 중개자가 JMS와 같은 엔터프라이즈-클래스enterprise-class 메시징 API나 AMQP와 같은 프로토콜들을 지원한다면, 레거시 시스템에서 보낸 메시지들을 소비하는 모바일과 웹 애플리케이션을 만들 수 있다.

이 책의 내용

이 책에서는 모바일 디바이스들과 웹 애플리케이션들을 위한 메시징 프로토콜을 소개한다.

1장, 소개

이 장에서는, 메시징 프로토콜의 개념과 모델, 메시지 표현을 소개한다. 메시징 프로토콜이 모바일과 웹 플랫폼에서 어떻게 사용되는지 설명하기 위해 2개의 애플리케이션을 만들어 볼텐데, 하나는 STOMP를 사용하고, 다른 하나는 MQTT를 사용한다. 또한, 두 애플리케이션은 두 개의 파트로 나뉘며(하나는 모바일 디바이스에서, 다른 하나는 웹 브라우저에서 실행된다), 서로 메시지를 사용하여 통신한다. 이 장은 두 예제 애플리케이션의 전체적인 디자인도 함께 설명한다.

2장, STOMP를 통한 모바일 메시징

이 장에서는, 단순한 텍스트 기반의 메시징 프로토콜인 STOMP를 소개한다. 여기서는 StompKit(STOMP를 위한 iOS 라이브러리)을 사용해서 디바이스에서 GPS 데이터를 보내고 텍스트 메시지를 받는 위치 추적 애플리케이션을 만들어 본다.

3장, STOMP를 통한 웹 메시징

이 장에서는, STOMP를 위한 자바스크립트 라이브러리인 stomp.js를 소개한다. 또한, 위치 추적 iOS 애플리케이션으로부터 메시지를 받고 수집된 GPS 데이터를 지도상에 표현하는 웹 애플리케이션도 함께 소개한다. 이 애플리케이션은 iOS 애플리케이션으로 텍스트 메시지를 보내기도 한다.

4장, STOMP 고급

이 장에서는, 2장과 3장의 애플리케이션 개발에서는 사용하지 않았던 STOMP의 고급 기능들을 소개한다. 이 고급 기능들은 메시징 애플리케이션들에서 항상 사용되지는 않는다. 하지만 애플리케이션이 점점 복잡해짐에 따라 점점 유용해지는 기능들이다.

5장, STOMP를 넘어서

이 장에서는, STOMP에서 제공되지는 않지만 STOMP 중개자들로부터 이용 가능한 기능들에 대해 다룬다. 이 기능들은 흔히 발생하는 이슈들을 해결해주고 코드 복잡도를 절감시켜주곤 한다.

6장, MQTT를 통한 모바일 메시징

이 장에서는, 바이너리 메시징 프로토콜인 MQTT를 소개한다. 이는 모바일이나 임베디드 디바이스들에서 데이터를 동보하는 데 적합한 프로토콜이다. 또한, 위치 추적 모바일 iOS 애플리케이션도 함께 살펴볼 것이다. 이 애플리케이션은 MQTT를 사용해서 디바이스의 움직임 데이터를 동보(MQTTKit 라이브러리를 사용)하고, 알림을 받게 되면 애플리케이션의 배경 색상을 변경한다.

7장, MQTT를 통한 웹 메시징

이 장에서는, 웹 소켓상에서 MQTT를 사용하는 웹 애플리케이션을 만들어 볼 것이다. 이는 디바이스의 움직임 데이터를 출력하기 위해 위치 추적 애플리케이션과 통신하며, 디바이스들이 색상을 변경하도록 알림을 보낸다.

8장, MQTT 고급

이 장에서는, 6장과 7장에서 사용되지 않았던 MQTT의 고급 기능들을 소개한다.

부록 A, ActiveMQ

이 부록에서는, 이 책의 STOMP 예제에서 사용되는 메시징 중개자인 아파치 ActiveMQ 메시징 중개자를 어떻게 설치하고 설정하는지에 대해서 설명한다.

부록 B, Mosquitto

이 부록에서는, Mosquitto 중개자를 어떻게 설치하고 설정하는지와 더불어 Mosquitto 중개자의 커맨드 라인 도구를 함께 소개한다. Mosquitto는 이 책에서 MQTT 메시지를 주고받는 데 사용된다.

이 책은 순서대로 읽도록 구성되었지만, 여러분의 경험에 따라 몇몇 장은 건너 뛰어도 된다. 1장은 이 책에서 다뤄지는 개념들을 소개한다.

만약 모바일 애플리케이션에 관심이 있다면, 2장과 6장에 집중하면 된다. 여기서는 모바일 디바이스를 위한 2개의 서로 다른 메시징 프로토콜을 소개하고 있다. 만약 웹 애플리케이션을 작성하는 데 관심 있다면 3장과 7장을 살펴보면 된다.

STOMP 프로토콜에 주로 관심있다면, 2, 3, 4, 5장을 보고, MQTT에 집중하고자 한다면 6, 7, 8장을 읽으면 된다.

대상 독자

이 책은 STOMP와 MQTT 메시징 프로토콜을 소개하고 있으며 이들에 대한 경험이 없는 독자를 전제로 하고 있다. 각 플랫폼의 클라이언트들이 제공하는 프로토콜을 다루기 위한 API들이 서로 다를 수도 있지만, 근본적인 개념은 같다. 각각의 프로토콜을 위해 두 개의 서로 다른 라이브러리를 살펴볼 것이다. 하나는 iOS를 위한 오브젝티브-C 라이브러리고, 하나는 웹 애플리케이션을 위한 자바스크립트 라이브러리다.

기본적인 프로그래밍 스킬이 필요하다. 이 책의 예제들은 서로 다른 플랫폼에서 실행되며, 각각에 적합한 프로그래밍 언어들을 사용하고 있다.

iOS에서 동작하는 모바일 애플리케이션은 오브젝티브-C로 작성되어 있다. 그래픽 구현을 위해 Xcode와 인터페이스 빌더Interface Builder에 대한 최소한의 지식을 필요로 하지만, 책에서의 모든 변경 사항은 하나씩 차례대로 설명한다.

웹 애플리케이션은 자바스크립트를 사용한다. 웹 애플리케이션을 인터랙티브interactive하게 만들고 페이지 요소들을 조작하기 위해 jQuery[05]를 사용하고 있지만, 메시징 코드는 어느 자바스크립트 프레임워크와도 독립적이다.

이 아이콘은 일반적인 주석을 의미한다.

이 아이콘은 팁이나 제안을 의미한다.

이 아이콘은 경고나 주의를 의미한다.

예제 코드

이 책의 모든 내용은 저작자 표시-비영리-변경 금지 4.0 인터내셔널 라이선스Attribution-NonCommercial-NoDerivatives[06] 아래에 있으며, GitHub 이슈 트래커[07]를 통해 기능 요청, 오류 문구 정정, 개선 등의 기여를 할 수 있다. 모든 문구와 예제를 저작자 표시를 필요로 하는 라이선스를 포함하면 재사용할 수 있다. 자세한 내용은 전

05 http://jquery.com/
06 http://bit.ly/cc-nc-nd
07 http://bit.ly/mwm-issues

체 라이선스를 통해 확인하자.

예제 코드 등 보충 자료는 https://github.com/mobile-web-messaging/code/에서 내려받을 수 있다.

이 책은 예제를 실행하는 데 필요한 모든 코드와 iOS 애플리케이션의 사용자 인터페이스를 설정하는 데 필요한 모든 설명을 담고 있다.

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. 무료로 업데이트되는, 전자책 전용 서비스입니다.

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

1 | 소개

이번 장에서는 메시징 프로토콜messaging protocol의 주요 개념들을 소개한다. 이후에는 이 프로토콜들이 모바일과 웹 플랫폼에서 어떻게 쓰이는지 설명하기 위해 직접 애플리케이션을 만들 것이다. 하나는 STOMP를 위한 것이고, 하나는 MQTT를 위한 것이다. 일단 이번 장에서는 두 애플리케이션의 전체적인 설계를 훑어보고 이어지는 장들에서 작성해보자.

1.1 메시징 개념

앞에서 2개의 문장과 5개의 개념으로 메시징 프로토콜을 소개했다.

- 애플리케이션은 **중개자**를 통해 **목적지**로 **메시지**를 **공급**한다.
- 애플리케이션은 메시지를 **소비하기** 위해 같은 목적지를 구독한다.

이제 이 5개의 개념을 정의해보자. 이들은 [그림 1-1]에서도 표현되고 있다.

메시지Message

애플리케이션 간에 교환되는 데이터

목적지Destination

메시지를 교환하는 데 사용되는 주소의 한 형태

공급자Producer

메시지를 목적지로 보내는 애플리케이션

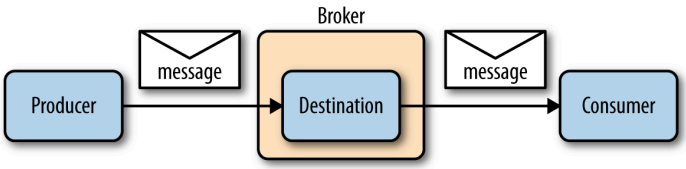
소비자Consumer

메시지를 목적지로부터 소비하는 애플리케이션

중개자Broker

공급자로부터의 메시지를 다루고 목적지에 따라 소비자에게 메시지를 전달하는 하나의 독립된 개체

[그림 1-1] 메시징 개념



메시징의 간단함이 마치 속임수 같을지도 모르겠다. 하지만 이 간단함 덕분에 우리는 메시징을 매우 유용한 방식으로 사용할 수 있다.

메시징 프로토콜이나 모델에 따라 공급자producer는 전달자sender나 게시자publisher로 불리기도 한다. 비슷하게 소비자consumer는 수신자receiver나 구독자publisher로 불릴 수 있다. 이 책에서는 일반적 용어인 공급자와 소비자를 사용할 것이다.

메시징의 주요 측면 중 하나는 참여자 간 결합도가 낮다는 것이다. 공급자와 소비자는 서로에 대해 아는 것이 없다. 임의의 애플리케이션이 메시지를 공급할 때 메시지가 언제 어디에서 소비되는지 전혀 알지 못한다. 메시지를 받는 수신자는 하나일 수도 있고 여럿일 수도 있다. 만약 아무도 관심 없는 경우라면 메시지가 전혀 소비되지 않을 수도 있다.

마찬가지로 애플리케이션이 메시지를 소비할 때도 직접적인 통신을 하는 게 아니기 때문에 어떤 애플리케이션이 보낸건지 모른다. 메시지에 공급자 애플리케이션에 대한 정보가 담겨있을 수도 있지만, 이는 필수가 아니다(대개의 경우 불필요하다).

심지어 공급자와 소비자는 동시에 온라인일 필요도 없다. 공급자는 메시지를 보내고 종료될 수 있다. 이 메시지는 소비자가 같은 목적지를 구독할 때까지 중개자에

의해 보관된다. 그리고 이 구독 시점에 중개자는 소비자에게 메시지를 전달한다.

공급자와 소비자는 자신들이 연결될 중개자를 알아야 한다. 하지만 꼭 같은 중개자에 연결될 필요는 없다. 여러 개의 중개자들이 하나의 클러스터로 구성될 수 있고, 메시지는 소비자에게 최종 전달되기 전에 이 중개자들을 거쳐갈 수 있다.

1.2 메시징 모델

메시징 모델은 메시지가 어떻게 공급자와 소비자 사이를 이동하는지 설명한다.

주요 메시징 모델로는 다음 2가지가 있다.

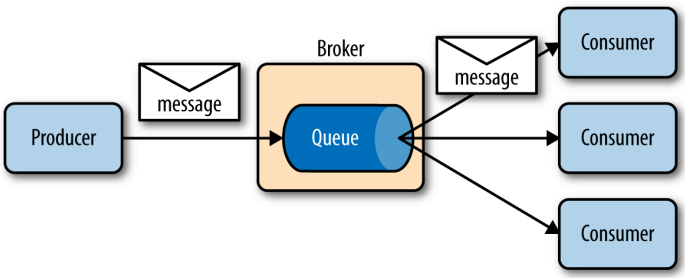
- 점대점Point-to-Point
- 게시/구독Publish/Subscribe

1.2.1 점대점

점대점Point-to-Point 메시징 모델에서는 공급자에 의해 보내진 메시지가 1명의 소비자에게만 전달된다.

공급자는 메시지를 큐queue에 의해 식별되는 목적지로 보낸다. 이 큐는 0명, 1명, 다수의 소비자가 구독(또는 연결)할 수 있으며, 메시징 중개자는 1명의 소비자에게만 메시지를 전달한다. [그림 1-2]에서 표현된 것처럼 공급자가 큐에 메시지를 보낼 때 구독하고 있는 소비자들 중 1명만이 메시지를 받는다.

[그림 1-2] 점대점 토폴로지(topology) 다이어그램



이 모델은 '일대일 메시징 모델one-to-one messaging model'이라고 불리기도 한다(1명의 공급자에 의해 메시지가 큐로 전달되고, 이를 받는 소비자는 오직 1명이다).

만약 큐에 연결된 소비자가 없다면 중개자는 소비자가 구독할 때까지 들어온 메시지를 보관하고 있다가 이 소비자에게 메시지를 전달한다. 몇몇 중개자들은 메시지가 큐에 보관되고 나서 일정 시간이 지나면 이를 만료시켜 버린다. 이는 소비자가 너무 오래된 메시지를 받지 않게 하는데 유용하다.

점대점 모델은 오직 1명의 소비자만이 메시지를 처리해야 하는 경우에 주로 쓰인다. 이때의 큐는 메시지 처리를 여러 클라이언트들에게 부하분산시키고 오직 1명의 클라이언트만 메시지를 수신할 수 있도록 보장해준다.

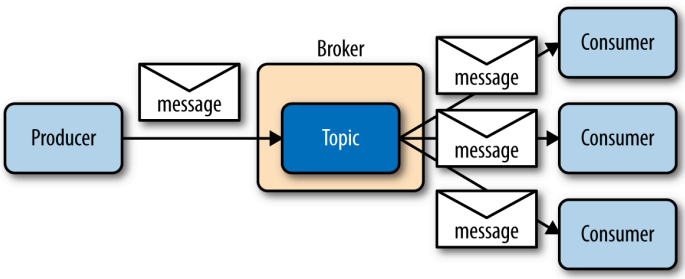
1.2.2 게시/구독

'게시/구독 메시징 모델(publish/subscribe messaging model, 줄여서 pub/sub 이라고도 부른다)'에서는 공급자의 메시지가 여러 명의 소비자에게 전달된다.

공급자는 토픽topic에 의해 식별된 목적지로 메시지를 전달한다. 이 토픽을 구독하는 소비자는 0명이거나 여러 명일 수 있고, 메시징 중개자는 들어온 메시지를 모든 구독자들에게 전달한다. 토픽에 연결된 소비자가 아무도 없는 경우, 중개자는 들어온 메시지를 보관하지 않는다. [그림 1-3]에서 표현된 것처럼 공급자가 토픽에 메

시지를 보내면 이 토픽에 연결된 모든 소비자가 메시지를 받게 된다.

[그림 1-3] 게시/구독 토폴로지 다이어그램



또한, 이 모델은 '일대다 메시징 모델one-to-many messaging model'이라고도 불린다(메시지를 보내는 1명의 공급자가 있고, 이 메시지를 받는 여러 명의 수신자가 있다).

이 모델에서는 메시지가 토픽에 전달되는 것을 가리켜 수신자들에게 동보broadcasting된다고 표현한다. 모든 수신자가 메시지를 전달받기 때문이다. 몇몇 프로토콜에서는 '지속형 구독자durable subscriber'라는 용어를 사용한다. 만약 소비자가 지속형 구독자로 토픽을 구독하면 소비자가 오프라인인 경우 메시지를 보관해두었다가 온라인이 되었을 때 메시지를 보내준다.

게시/구독 모델은 특히 변경사항을 보내는 데 적합하다. 공급자는 토픽에 변경된 데이터들을 보내고 토픽을 구독하는 소비자들은 이를 통지받는 것이다.

1.3 메시지 표현

공급자와 소비자는 메시지를 통해 정보를 주고받는다.

[그림 1-4]에서 표현된 것처럼 메시지는 3개의 분리된 데이터 조각으로 구성된다. 바로 목적지destination, 헤더headers, 바디body다.

[그림 1-4] 메시지 다이어그램

공급자가 목적지로 메시지를 보낼 때 목적지의 이름이 메시지 안에 들어간다. 이 정보를 통해 소비자는 전달받은 메시지가 어떤 목적지의 것인지 알 수 있다. 이는 소비자가 여러 목적지를 구독하고 있는 경우 메시지를 식별하려고 할 때 특히 유용하다.

메시지는 헤더도 가지고 있다. 메시징 프로토콜은 메시지에 메타데이터metadata를 추가할 때 헤더를 사용한다. 소비자에게 읽히는 이 메타데이터는 메시지에 맥락 정보들을 더한다. 메타데이터의 예로는 중개자가 메시지를 고유하게 구별할 수 있게 하는 메시지 식별자, 공급자가 보낸 날짜와 시간을 나타내는 타임스탬프, 메시지 전송이 한 번 실패한 후에 다시 보내진 것인지 여부를 나타내는 재전송 여부 등이 있다. STOMP와 같은 몇몇 메시징 프로토콜들은 프로토콜에 의해 정의된 헤더에 더해 애플리케이션만의 고유한 헤더들을 공급자가 추가할 수 있게 허용하고 있다. MQTT 같은 나머지 프로토콜들은 애플리케이션의 특수한 헤더를 허용하지 않는다. 이런 경우에는 애플리케이션의 특수한 헤더들을 메시지의 페이로드payload에 넣어야만 한다.

마지막으로 메시지는 공급자와 소비자 간에 교환되는 데이터를 포함하는 바디나 페이로드를 선택적으로 담을 수 있다. 바디의 형태는 메시징 프로토콜에 따라 다르다. 몇몇은 STOMP처럼 텍스트 형식으로 정의하고, 몇몇은 MQTT 같은 바이너리 형식으로 정의한다. 페이로드는 내용을 담은 불투명한 영역이다. 중개자는 메시지를 전달할 때 이를 읽어보지도 수정하지도 않는다.

정보를 전달하는 대부분의 경우 오로지 메시지의 바디만을 사용할 것이다. 다양한 형태(JSON 문자열, 단순 문자열, 소수의 배열 등)를 이용해서 말이다. 하지만 프로토콜이 허용만 해준다면 바디의 메타데이터(내용의 종류나 길이 등)를 알려주거나 중개자의 기능들을 작동시키기 위해 메시지에 부가적인 헤더들을 추가할 것이다.

1.4 예제

메시징 프로토콜이 모바일과 웹 플랫폼에서 어떻게 쓰이는지 보기 위해 이 책에서는 2개의 애플리케이션 세트를 만들어 볼 것이다. 각 세트는 iOS 애플리케이션과 웹 애플리케이션으로 구성된다. 첫 번째 애플리케이션은 STOMP를, 두 번째 애플리케이션은 MQTT를 사용한다.

1.4.1 STOMP를 이용한 위치 추적 애플리케이션

트럭을 이용하여 고객에게 물건을 배달하는 회사에서 일하고 있다고 가정해보자. 각각의 트럭들은 물건을 배달하는 일을 담당하고, 운전사들은 본사로부터 주문을 받는다. 모든 트럭들을 효과적으로 관리하기 위해 회사는 트럭의 위치를 추적하여 필요에 따라 주문을 조정하고 싶어한다.

이 요구를 만족시키기 위해 이제부터 위치 추적이라는 이름의 간단한 애플리케이션을 만들어 보고자 한다([그림 1-5]와 같은 모습이 될 것이다).

먼저 "트럭"은 iOS 애플리케이션을 통해서 GPS 센서로 얻은 디바이스의 위치 데이터를 동보broadcast하고 본사로부터 받은 텍스트 메시지를 확인한다. "본사"는 웹 애플리케이션을 통해서 모든 트럭의 위치를 지도에서 확인할 수 있다. 또한, 트럭에 텍스트 메시지를 보낼 수도 있다.

- 2장에서는 STOMP 프로토콜을 이용하여 GPS 데이터를 보내고 문자메시지를 받는 "위치 추적" iOS 애플리케이션을 작성해 볼 것이다.

- 3장에서는 STOMP 프로토콜을 이용하여 웹 브라우저에서 웹소켓Web Sockets으로 GPS 데이터를 받고 iOS 디바이스들에 텍스트 메시지를 보내는 웹 애플리케이션을 만들어 볼 것이다.

이 애플리케이션에서 사용되는 STOMP 프로토콜을 소개하기에 앞서 애플리케이션의 메시징 모델과 메시지를 어떻게 주고받을지에 대해 결정해 볼 수 있다.

위치 추적 애플리케이션을 위한 메시징 모델

애플리케이션에서 사용되는 목적지는 2개가 된다. 하나는 iOS 디바이스의 위치 정보를 동보하는 게시/구독 모델과 함께 쓰이고, 하나는 텍스트 메시지를 위한 점대점 모델과 함께 쓰인다.

- device.XXX.location은 GPS 위치정보를 동보하는 토픽이다. 여기서 XXX는 디바이스의 식별자가 된다(게시/구독 모델).
- device.XXX.text는 간단한 텍스트 메시지를 받는 큐다(점대점 모델).

토픽은 여러 소비자들이 정보를 수신할 수 있게 해주기 때문에 GPS 데이터를 보내는 데 사용된다. 한편, 큐는 단일 디바이스가 메시지를 소비할 수 있도록 텍스트 메시지를 다루는 데 사용된다.

[그림 1-5] 두 개의 디바이스 AAA와 BBB, 두 개의 웹 애플리케이션을 나타내는 다이어그램

위치 추적 애플리케이션이 실행되고 있으며 고유 식별자 XXX에 의해 식별되는 각 디바이스는 다음과 같이 설명할 수 있다.

- device.XXX.location 토픽에 대한 메시지 공급자
- device.XXX.text 큐의 유일한 메시지 소비자

반대로 웹 애플리케이션은 다음처럼 설명된다.

- device.XXX.location 형태의 모든 토픽에 대한 메시지 소비자
- device.XXX.text 형태의 모든 큐에 대한 메시지 공급자

위치 추적 애플리케이션을 위한 메시지 표현

교환되는 메시지에는 두 가지 종류가 있다.

- device.XXX.location 토픽에 의해 교환된 GPS 데이터를 표현하는 메시지
- device.XXX.text 큐에 의해 교환된 주문을 표현하는 메시지

iOS 애플리케이션에서는 메시지 페이로드에 JSON 형태의 GPS 데이터를 보낸다([예제 1-1] 확인).

[예제 1-1] 위치 메시지 페이로드

```
{
"deviceID": "BBB", ❶
"lat": 48.8581, ❷
"lng": 2.2946, ❸
"ts": "2013-09-23T08:43Z" ❹
}
```

❶ deviceID는 위치를 전송하는 디바이스에 대한 식별자다.

❷ lat은 위도를 나타내는 숫자다.

❸ lng은 경도를 나타내는 숫자다.

❹ ts는 위치정보가 기록된 시간을 나타내는 문자열이다(ISO 8601[01] 형식을 사용한다).

iOS 애플리케이션에서 소비되는 메시지는 다음과 같이 간단한 평문으로 표현된다.

[예제 1-2] 텍스트 메시지 페이로드

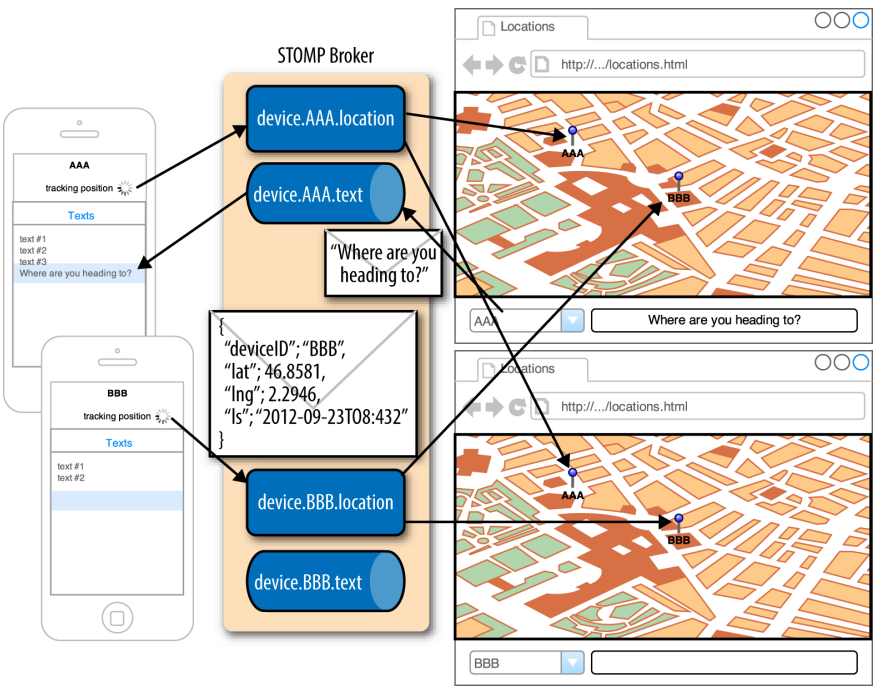
```
"Where are you heading to?"
```

01 역자주_국제 표준화 기구(ISO)에서 제정한 날짜와 시간의 표기에 관한 국제 표준 규격을 가리킨다(출처: http://ko.wikipedia.org/wiki/ISO_8601).

좀 더 현실적인 메시지라면 어느 본사로부터의 메시지인지를 나타내는 식별자 같은 정보를 더 많이 담고 있을 수 있다. 하지만 이번 예제에서는 평문을 사용하는 것만으로 충분하므로 당분간 계속 사용할 것이다.

지금까지 알아본 메시징 토폴로지와 데이터 표현을 통해 위치 추적 애플리케이션의 다이어그램이 [그림 1-6]처럼 개선될 수 있다.

[그림 1-6] 메시징 모델과 데이터 표현이 담긴 위치 추적 애플리케이션 다이어그램



1.4.2 MQTT를 이용한 움직임 추적 애플리케이션

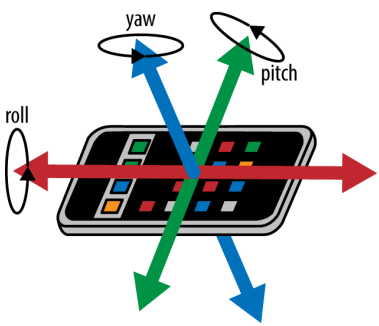
대부분의 모바일 디바이스들은 디바이스의 움직임을 추적할 수 있는 센서를 가지고 있다(사용자의 움직임도 어느 정도까지는 가능하다). 추가적인 센서들을 이용하면 사용

자의 건강 데이터(심박동수, 수화작용, 혈당수치 등)도 모니터링할 수 있다. 이러한 정보들은 중앙 애플리케이션에 보내져서 모니터링되다가 필요한 경우 사용자에게 경보를 보내줄 수 있다.

이 모델을 이용해 움직임 추적이라는 애플리케이션을 간단하게 작성해보며, MQTT 프로토콜에 대해 알아볼 것이다. iOS 애플리케이션은 디바이스의 움직임을 추적하다가 경보 메시지가 도착하면 이를 사용자에게 알리기 위해 배경색을 빨간색으로 바꾼다.

디바이스의 움직임은 [그림 1-7]에서 보듯 피치pitch, 롤roll, 요yaw라는 3가지 값으로 표현된다.

[그림 1-7] 디바이스의 움직임을 표현하는 피치, 롤, 요

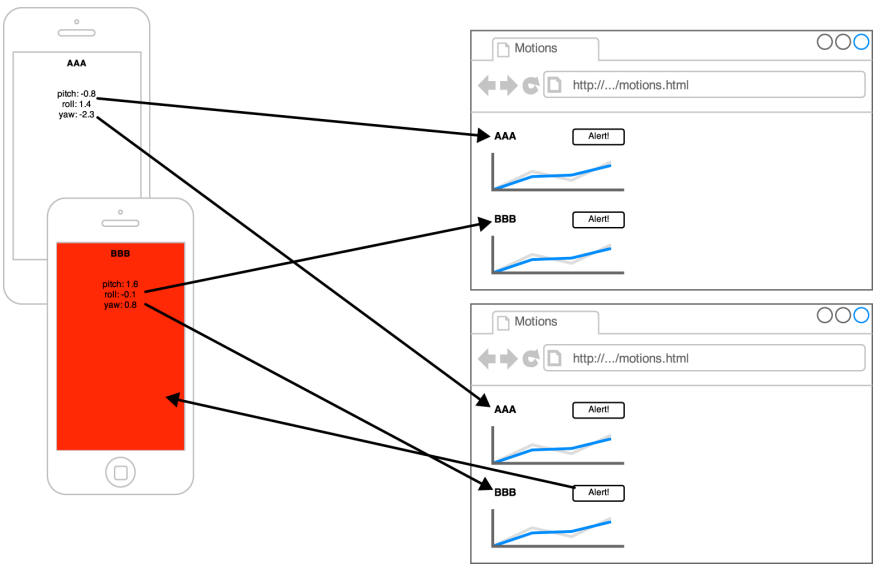


웹 애플리케이션은 동보된 모든 디바이스들의 움직임을 보여주고 이 디바이스들에게 경보 메시지를 보낼 수 있다([그림 1-8]).

- 6장에서는 MQTT 프로토콜을 이용하여 디바이스의 움직임을 전송하고 경보를 수신하는 "움직임 추적" iOS 애플리케이션을 작성할 것이다.
- 7장에서는 MQTT 프로토콜을 이용하여 웹소켓으로 디바이스들의 움직임 데이터를 수신하고, 이를 보여주는 웹 애플리케이션을 작성할 것이다. 또한, 이

웹 애플리케이션은 움직임 데이터를 보내는 어느 디바이스에도 경보 메시지를 전달할 수 있다.

[그림 1-8] 두 개의 클라이언트(AAA, BBB)와 이들을 모니터링하는 두 개의 웹 애플리케이션



움직임 추적 애플리케이션을 위한 메시징 모델

이 애플리케이션에서는 게시/구독 모델과 함께 2개의 목적지를 사용한다.

- /mwm/XXX/motion(여기서의 XXX는 디바이스 식별자)는 디바이스의 움직임 데이터를 동보하기 위한 토픽이다(게시/구독 모델).
- /mwm/XXX/alert는 주어진 디바이스로 경보 메시지를 교환하기 위한 토픽이다(게시/구독 모델).

움직임 추적 애플리케이션을 구동하는 각 디바이스들은 다음과 같이 설명될 수 있다.

- /mwm/XXX/motion 형태의 토픽으로의 메시지 공급자
- /mwm/XXX/alert 형태의 토픽으로부터의 메시지 소비자

반대로 웹 애플리케이션은 다음과 같이 설명된다.

- /mwm/XXX/motion 형태의 모든 토픽으로부터의 메시지 소비자
- /mwm/XXX/alert 형태의 모든 토픽으로의 메시지 공급자

MQTT는 오직 게시/구독 메시징 모델만을 지원한다. 이상적으로는 경보 목적지가 큐로 모델링(디바이스별로 하나의 큐)되는 게 더 좋다. 하지만 MQTT는 큐를 지원하지 않기 때문에 여기서는 각 디바이스별로 하나의 토픽을 두고 디바이스가 각각의 토픽을 구독하게 할 것이다.

움직임 추적 애플리케이션을 위한 메시지 표현

교환되는 메시지에는 두 가지 종류가 있다.

- 디바이스의 움직임 데이터를 표현하는 메시지(/mwm/XXX/motion 토픽들에서 교환된다)
- 경보를 표현하는 메시지(/mwm/XXX/alert 토픽들에서 교환된다)

움직임 추적 iOS 애플리케이션은 디바이스의 움직임 데이터를 바이너리 메시지로 보낸다. 이 메시지는 피치, 롤, 요 이렇게 3개의 64-비트 소수로 구성된다([예제 1-3]).

[예제 1-3] 디바이스 움직임 데이터 페이로드

```
<< 1.6 -0.1 0.8 >> ❶
```

❶ 이 메시지는 피치, 롤, 요를 나타내는 3개의 64-비트 소수들로 구성되어 있다.

움직임 추적 iOS 애플리케이션에서 소비되는 경보 메시지는 색상을 표현하는 간

단한 평문으로 표현된다. 애플리케이션에서는 이 페이로드를 이용해서 사용자에게 경보하기 위해 배경색상을 일시적으로 바꿀 것이다([예제 1-4]).

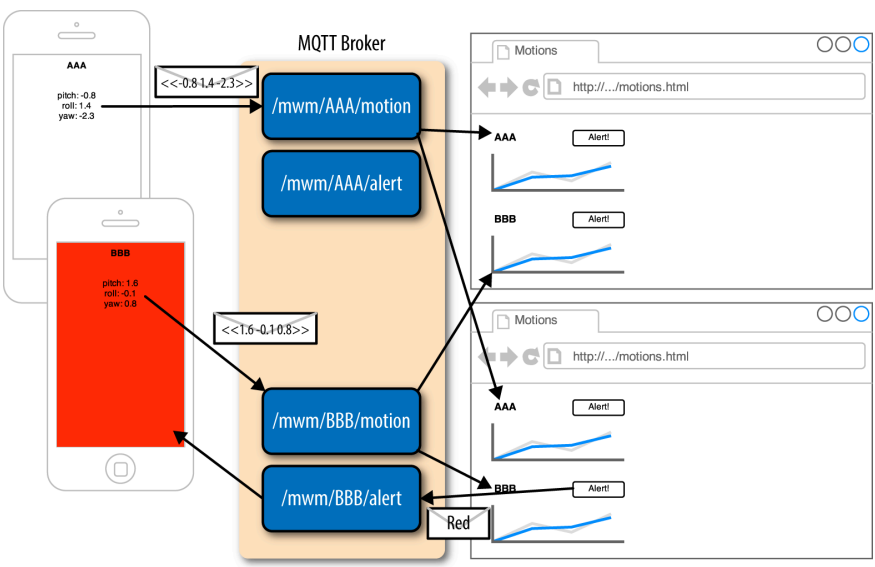
[예제 1-4] 경보 메시지 페이로드

```
"red" ❶
```

❶ 이 메시지는 색상 이름을 담고 있는 문자열로 구성되어 있다.

지금까지 알아본 메시징 토폴로지와 데이터 표현을 통해 움직임 추적 애플리케이션의 다이어그램이 [그림 1-9]처럼 개선될 수 있다.

[그림 1-9] 메시징 모델과 데이터 표현이 담긴 움직임 추적 애플리케이션 다이어그램



1.5 요약

이번 장에서는 메시징 프로토콜에 대해서 알아보고, 요청/응답 프로토콜request/reply

protocol과는 어떻게 다른지에 대해서 배워보았다. 그리고 두 개의 메시징 모델(점대점, 게시/구독)과 메시지의 구성 요소(목적지, 헤더, 페이로드)에 대해서도 알아보았다.

더불어 다음 장들에서 구현하게 될 2개의 애플리케이션(STOMP를 이용하는 위치 추적 애플리케이션과 MQTT를 이용하는 움직임 추적 애플리케이션)에 대해서도 간단히 훑어보았고, 앞으로 사용하게 될 메시징 모델과 표현을 만들어 보았다.

다음 장에서는, 위치 추적 iOS 애플리케이션을 바로 만들어보겠다. 만약 여러분이 MQTT를 배우는 것에 더 관심이 있다면 바로 6장으로 넘어가서 움직임 추적 애플리케이션 작성을 시작해보기 바란다.

다음 4개 장에서는 애플리케이션을 만드는 데 STOMP 프로토콜을 사용한다(이 책에서는 최신 프로토콜 버전인 STOMP 1.2[01]를 사용한다). STOMP는 단순한 텍스트 기반text-based의 메시징 프로토콜messaging protocol이며, 어느 플랫폼에서든 경량의 메시징 애플리케이션을 개발하는 데 적합하다. 또한, 상호운용 가능한interoperable 연결 형식을 제공하여 어떤 클라이언트든지 간에 임의의 중개자와 통신할 수 있게 해준다. 프로토콜의 단순함은 이러한 클라이언트와 중개자 사이의 상호 운용성을 쉽게 구축할 수 있게 도와준다. STOMP는 목적지의 시맨틱semantics[02]을 정의하지는 않는다(이는 여러분이 사용하는 STOMP 중개자에 따라 다를 수 있다).

STOMP 중개자들 사이에서 공유되는 관습들이 있긴 하지만(큐를 사용하기 위해서는 /queue/를, 토픽을 사용하기 위해 /topic/을 목적지의 접두사로 사용한다), 중개자들이 제공하는 문서를 통해 점대점 모델과 게시/구독 중 어떤 메시징 모델을 사용하고, 또 어떻게 사용하는지를 확인해야 한다.

STOMP는 몇 개의 구문 분석 규칙과 함께 텍스트에 기반을 두고 있으며, 이는 텍스트를 읽고 작성하고 네트워크 연결을 맺을 수 있는 어느 플랫폼에서든 STOMP를 쉽게 사용할 수 있게 도와준다. 그러나 텍스트 기반의 프로토콜은 가장 효율적인 프로토콜은 아니다. 바이너리 기반의 프로토콜에 비해 더 많은 네트워크 대역폭과 메모리를 필요로 하기 때문이다. 만약 여러분의 애플리케이션이 이런 제약과 별로 상관이 없다면, STOMP는 사용하기 좋은 프로토콜이 될 것이다.

---

01 http://bit.ly/STOMPspec

02 역자주_ 목적지로 사용된 데이터의 문법이나 규칙이 어떤 의미로 해석할지 결정하는 것을 말한다.

STOMP 클라이언트를 사용하기 전에 메시지를 중개하기 위한 중개자가 설치되고 설정되어 있어야 한다. 이 책에서는 Apache ActiveMQ가 사용되며, 부록A에서 설치 및 설정 방법을 확인할 수 있다. ActiveMQ가 시작되면, 61613 포트를 통해 STOMP 연결을 맺을 수 있다.