# 이동 단말기의 메모리 제약을 해결하기 위한 자료 구조*

김찬우[0] 김재훈
아주대학교 정보통신전문대학원 정보통신공학과
{chanux, jaikim}@madang.ajou.ac.kr

# Data Structures for Mobile Devices

Chan-Woo Kim[0] Jai-Hoon Kim
Graduate School of Information and Communication, Ajou University

**요 약**

이동 통신 기기의 보급이 급증함에 따라 많은 모바일 응용들이 생겨나고 있다. 모바일 환경은 기존 PC 환경과는 달리 제한된 리소스를 제공하고 있기 때문에 어플리케이션을 제작하는 데에 많은 제약이 따르고 있다. 가장 큰 제약이라 할 수 있는 것은 제공되는 메모리의 양과 프로세서의 속도라 할 수 있다. 이 두가지는 서로 상반되는 특성을 가지고 있다. 연산 속도를 빠르게 하려면 보다 많은 메모리의 양이 필요로 하게 되고, 메모리를 절약하려면 연산 속도가 더 늦어지는 것이 일반적인 경우이다. 본 논문에서 다루는 것은 이러한 제한된 환경에서 어떻게 이러한 문제들을 해결하느냐에 대한 것이다. 본 논문에서는 메모리가 부족한 이동 단말기를 위한 어플리케이션 작성시 메모리 절약을 위한 자료구조 설정 방법의 예를 설명하고 메모리 사용량을 분석하였다.

## 1. 서 론

이동 통신은 최근 핸드폰을 비롯한 다양한 이동 통신 기기의 보급으로 인하여 그 중요성이 증대되고 있다. 이에 따라 수많은 이동 통신 기기(휴대용 정보기기)용 프로그램이 생겨나고 있다.

이동 통신 기기는 일반 PC와는 달리 다양한 디바이스와 다양한 OS를 가지고 있다. 게다가 PC에 비해 메모리 크기 또는 처리 속도에 있어서 다소 제한적인 환경을 가지고 있다. 처리 속도 문제라든지, 메모리의 양에 있어서는 PC에 비하여 상당히 제한적인 면이 있다. 일반적으로는 연산 속도의 증대를 위해선 보다 많은 메모리를 사용하게 되고, 메모리 사용량을 감소시키긴 위해선 연산 처리 속도가 느려지게 된다. 이동 단말기 및 어플리케이션에 따라 제한된 자원(CPU, 메모리)dmf 이용하여 사용자의 만족도를 극대화할 수 있는 방법이 필요하다. 본 논문에서는 메모리 크기의 제약이 많은 이동 단말기를 위한 어플리케이션을 작성할 때, 메모리 절약을 위해 자료구조를 어떻게 설정할 지에 대한 예를 설명하고 메모리 사용량을 분석하였다.

## 2. 관련연구

이동 통신 기기는 PC와는 달리 다양한 디바이스와 OS를 가지고 있다. 개발자는 개발하려는 어플리케이션을 각각의 플랫폼에 맞추어야 한다. 그러나 Java는 플랫폼에 독립적인 언어이므로, Java를 이용하여 어떤 단말기를 위한 어플리케이션을 개발하였다고 한다면, 다른 종류의 단말기로 그 어플리케이션을 옮기기 위해서 약간의 수정만으로도 그것이 가능하다. 이것이 Java를 많이 쓰는 이유 중의 하나일 것이다.

* 본 논문은 정보통신부에서 지원하는 대학기초연구지원사업으로 수행되었음. (2001-103-3)

Java 2 Platform은 [그림 1]에 나와있는 것처럼 J2EE, J2SE, J2ME로 나뉘어져 있다. J2EE는 서버를 위한 것이라 할 수 있고, J2SE는 일반 데스크탑을, 그리고 J2ME는 Embedded 장비를 위한 것이다. 특히 J2ME에서는 다양한 하드웨어를 분류하여 지원하기 위하여 Configuration과 Profile 이라는 개념을 도입하였다.

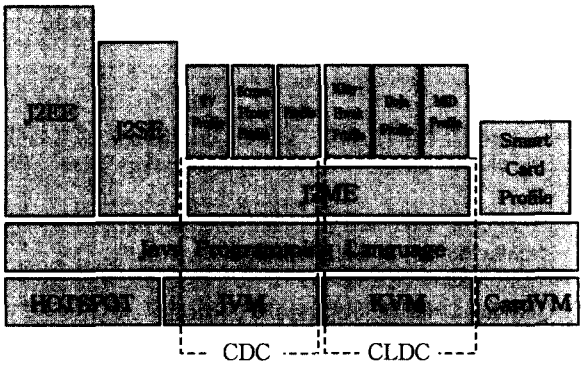


그림 1. Java 2 Platform

J2ME는 CDC (Connected Device Configuration)와 CLDC (Connected Limited Device Configuration)으로 나 Configuration을 나누고 있는데 그 명세는 다음과 같다.

- profile이 올라갈 수 있는 Java 2 platform의 core technology에 대한 명세
- Java 언어에 대한 명세 : 부동 소수점 지원 여부
- 가상 머신에 대한 명세 : KVM, pJava, eJava
- Core API에 대한 명세 : java.* 패키지

Profile은 특정 산업이나 디바이스의 환경에 따라 그에 맞게 정의된 자바 API의 명세라고 할 수 있다. CDC와 CLDC의 Profile은 다음과 같다.

- CDC의 profile : TV, Screen Phone, Car 등
- CLDC의 profile : MID, DoJa, KittyHawk 등

KVM(Kilobyte Virtual Machine)은 가상머신으로 기존의 가상 머신 (pJava, eJava)는 JVM의 서브셋을 목표로 설계되었으나, KVM은 작은 메모리 풋프린트를 가진 가상머신을 재설계한 것으로 그 크기는 40KB 정도이다.

CLDC는 KVM을 기반으로 하여 mobile, personal, connected 장치를 위한 configuration으로써 크기가 작고 용량과 속도가 제한된 자원을 가진 장치에서 자바를 사용 가능하도록 가상 머신과 코어 자바 라이브러리를 재정의 하였으며, JSR-30이라는 표준 규격을 정의하였다.

CLDC에 설계된 디바이스의 환경은 다음과 같다

- 128~512KByte의 메모리 여유 공간
- 16/32Bit 프로세서
- 저전력 소모, 주로 배터리 이용
- 제한된 네트웍 대역폭
- 대량 생산이 가능한 장치

다음은 CLDC 명세서에서 정의하는 것들이다.

- 자원이 극도로 제한된 환경에서 작동할 수 있는 가상머신(KVM)의 특징
- 코어 자바 라이브러리 (java.*로 시작되는 패키지)
- 네트워킹과 입출력 (generic connect framework)
- 국제화
- 응용프로그램의 설치 및 관리

여기에서 정의되지 않는 것들은 Profile에서 정의된다. 그리고 다음은 CLDC가 기존의 J2SE와 다른 점에 대한 것이다.

- 부동 소수점을 지원하지 않는다
- finalization을 지원하지 않는다
- 에러 처리가 제한적이다
- JNI(Java Native Interface)를 지원하지 않는다
- 리플렉션을 지원하지 않는다
- 쓰레드 그룹과 데몬 쓰레드를 지원하지 않는다
- 사용자 정의 클래스 로더를 생성할 수 없다
- 약한 참조(weak reference)를 지원하지 않는다
- 클래스 검증 과정이 오프디바이스와 온디바이스로 2단계로 나뉘어져있다
- 클래스 파일 포맷이 다르고, 클래스 룩업, 클래스 로딩과 링킹 방법이 다르다
- Sandbox 보안 모델을 사용한다
- 전혀 다른 네트워킹 및 입출력 모델을 가지고 있다

CLDC의 profile에는 KittyHawk, MID, DoJa 등이 있는데, 본 논문에서 구현한 프로그램은 KityHawk와 관련된 것이므로 KittyHawk에 대해 알아보자.

KittyHawk은 CLDC의 Profile로써 200KByte 이하의 적은 메모리를 사용하도록 설계되어있다. KittyHawk을 이용하는 모든 응용 프로그램은 KHApp 또는 KH class로부터 시작되고, 저수준의 세밀한 graphic 제어를 위해 Canvas class가 사용되고, 고수준의 컴포넌트를 사용하여 응용 프로그램 제작할 경우 Screen class가 이용된다.

## 3. 구 현

본 장에서는 ez-Java를 이용하여 개발한 게임을 예제로 하여 메모리 사용 관련하여 분석을 해보았다.

- 구현한 하드웨어 환경 (LG i-book 모델)
  - Qualcomm MSM 3100
  - 4MB FlashMemory / 512KB StaticRAM
  - 128x128 pixel, 2bit gray Scale LCD
  - network speed : 64Kbps

- 소프트웨어 환경
  - KVM code size : 230 KBytes
  - KVM heap space : 108 KBytes
  - Java Application Storage : 64 KBytes x 10
  - Java 2 Platform Micro Edition, CLDC profile
  - KittyHawk User Interface API
  - Font : 한글 10x12, 영문 5x12
  - HTTP 1.1 구현 : JAM과 JAR 파일 다운로드

- 구현 프로그램 설명

간단한 퍼즐 게임을 구현하였다. 방향키를 누르면 누른 방향으로 장애물이 나타나는 곳까지 공이 움직이게 된다. 공을 네모 모양의 목표지점까지 움직이게 되면 게임을 종료하게 된다.

[그림 3]은 게임의 시작, 종료직전, 종료를 나타내는 그림이다.

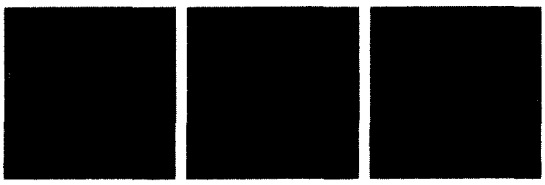

그림 3. 예제 프로그램 실행 화면

이 프로그램은 일반 PC에서 구현하는 것과는 다소 차이가 있는 방법으로 구현되었다. 트레이를 표현하기 위해서 그냥 단순한 방법으로 구현하려면 [그림 4]과 같은 구조로 배열에 저장하게 될 것이다.

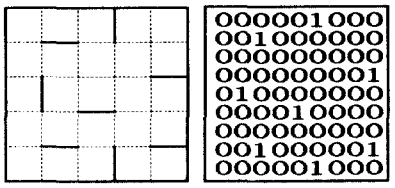


그림 4. 트레이의 구성 1

위와 같이 구성을 할 경우에는 구현하기 편하다는 장점이 있다. 그런데 이 게임은 보통 30개의 트레이로 구성 되어서 한 트레이가 끝나면 또 다음 트레이가 나오도록 되어있다. 하나의 트레이를 구성하는 이차원 배열만 고려한다면 5×5 크기의 트레이를 표현하기 위해서 (5×2-1)×(5×2-1)=81 byte를 차지하게 된다. 트레이의 개수를 총 30개라고 한다면 81×30=2430byte를 필요로 하게 된다.
그러나 [그림 5]와 같이 트레이를 구성하게 되면 메모리를 상당히 절약할 수 있게 된다.

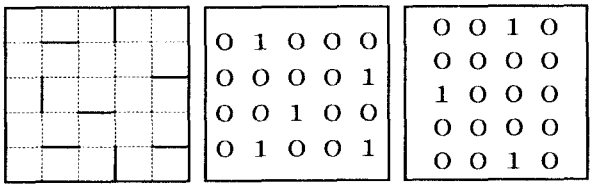


그림 5. 트레이의 구성 2

[그림 5]에서 첫 번째 그림은 트레이를 나타내는 것이고 두 번째 그림은 가로 벽을 나타내는 것, 세 번째 그림은 세로 벽을 나타낸다. 이 경우엔 5×5 크기의 트레이를 표현하기 위해서 5×(5-1)+(5-1)×5=40byte 이므로 총 30개를 나타내기 위해선 40×30=1200byte가 되어 [그림 4]에서의 것 보다 절반 이상의 메모리를 줄일 수 있게 된다.

$n \times n$ 크기의 트레이 m개를 표현하기 위해서는 첫 번째 방법으로는 $m(2n-1)^2$byte가 필요하게 된다. 그리고 두 번째 방법으로는 $2mn(n-1)$byte가 필요하게 된다. 따라서 두 번째의 방법을 이용하면 첫 번째의 방법보다 $m(2n^2-2n+1)$byte만큼 절약할 수 있게 된다.

[표 1]은 m=30일 때 n의 값에 따라 달라지는 배열의 개수를 [그림 4]의 방법과 [그림 5]의 방법으로 비교한 것으로 그래프는 [그림 6]에 나타나있다.

| n | 4 | 5 | 6 | 7 |
|---|---|---|---|---|
| 그림 4 | 1470 | 2430 | 3630 | 5070 |
| 그림 5 | 720 | 1200 | 1800 | 2520 |

표 1. m=30일 경우 필요한 배열의 개수

[표 2]은 n=5일 때와 m의 값에 따라 달라지는 배열의 개수를 [그림 4]의 방법과 [그림 5]의 방법으로 비교한 것으로 그래프는 [그림 7]에 나타나있다.

| m | 20 | 30 | 40 | 50 |
|---|---|---|---|---|
| 그림 4 | 1470 | 2430 | 3630 | 5070 |
| 그림 5 | 720 | 1200 | 1800 | 2520 |

표 2. n=5일 경우 필요한 배열의 개수

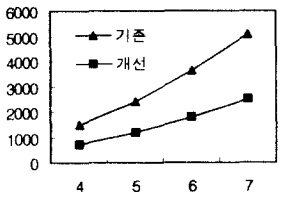


그림 6. [표 1]의 그래프

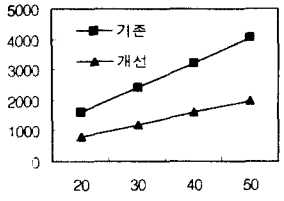


그림 7. [표 2]의 그래프

물론 구현하기는 [그림 4]의 것이 훨씬 더 간편하고, 수행속도도 더 빠를 수 있지만, 여기서 예로 든 LG i-book 모델은 한 프로그램당 64KByte로 제한이 되고 있기 때문에, 이를 극복하기 위해선 사용하는 메모리를 최대한 줄일 수 있어야 한다. 수행 속도의 문제에 있어서는 [그림 4]의 것이 더 우수한 면이 있으나 메모리에 드는 비용이 두배 차이나는 만큼이나 속도 차이가 있지는 않기 때문에 이 경우는 메모리를 줄여서 구현하는 것이 낫다고 할 수 있다.

그리고 메모리 절약을 위한 다른 방법으로는 굳이 필요한 경우가 아니라면 변수를 선언할 때 클래스의 멤버 변수로 하지 않는 것이 좋다. 일반 PC의 경우는 별 상관이 없겠지만 클래스가 초기화되고 소멸 될 때까지는 계속 메모리 상에 있기 때문에 적지 않은 메모리를 낭비하게 된다. 따라서 시기 적절하게 지역변수를 사용하면 메모리의 낭비를 막을 수 있다.

## 4. 결 론

본 논문에서는 모바일 환경에서의 프로그램 구현을 통하여 메모리의 낭비를 막을 수 있는 방법에 대해서 논의해보았다. 일반 PC에서의 프로그래밍에서는 메모리 제한에는 그리 크게 신경 쓸 필요가 없기 때문에 되도록 더 빠른 속도로 수행될 수 있게 하는 데에 중점을 두고 있지만 모바일 환경에서는 제한된 리소스를 감안해야 하므로 큰 차이가 나지 않는다면 되도록 메모리의 낭비를 줄이는 데에 주력해야 한다. 하지만 무조건 메모리의 낭비를 줄이는 데에만 신경쓸 것이 아니라, 메모리 사용량에 별 차이 없이 수행 속도를 향상시킬 수 있다면 당연히 수행 속도를 우선해야 할 것이다. 어떤 환경에서 프로그램을 작성하는 지에 따라 다르겠지만 각각의 환경이 제공하는 리소스의 조건에 따라 어떠한 것을 먼저 우선 하는 지를 먼저 생각한 다음 프로그램을 작성한다면 적은 리소스로 최대한의 결과를 낳을 수 있으리라 본다.

### 참고 문헌

[1] LG Telecom, "CLDC/KittyHawk Development's Guide", March 2, 2001
[2] Cay S. Horstmann, Gray Cornell, "Core Java 2 : Volume1-Fundamentals", 1st Edition, Sun microsystems, 1999
[3] Eric Giguère, "Java 2 Micro Edition", 1st Edition, New York: Wiley Computer Publishing, 2000
[4] Yu Feng and Dr. Jun Zhu, "Wireless Java Programming with J2ME", 1st Edition, Indiana : SAMS, 2001