Hanbit eBook

Realtime 98

안드로이드 애플리케이션의 성능 개선을 위한 스레드 관리

송무찬 지음

한빛미디어 Hanbit Media, Inc.

안드로이드 애플리케이션의 성능 개선을 위한 스레드 관리

송무찬 지음

한빛미디어 Hanbit Media, Inc.

안드로이드 애플케이션의 성능 개선을 위한 **스레드 관리**

초판발행 2015년 5월 13일

지은이 송무찬 / **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 978-89-6848-747-7 15000 / **정가** 12,000원

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

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

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_ **송무찬**

몇 년 전에 실리콘밸리에 정착해서 살고 있는 개발자입니다. 십 년 동안 일하며 대부분의 시간은 서버를 개발하면서 보냈고, 안드로이드 개발은 삼 년 전부터 시작했습니다. 서버를 개발할 당시에 이해하지 못했던 클라이언트의 고충(?)을 안드로이드를 개발하면서 이해하게 되었고, 현재는 안드로이드 앱을 개발하며 서버와 클라이언트가 쉽게 인터페이스할 수 있는 기술에 대해서 고민하고 있습니다.

안드로이드는 2008년 9월 23일 1.0을 발표한 이후로 매우 빠르게 성장하여 현재는 가장 많은 기기에서 사용하는 플랫폼이 되었습니다. 구글 플레이 앱스토어에 배포된 앱의 개수 또한 애플 앱스토어의 개수를 뛰어넘었으며, 안드로이드 앱은 안드로이드를 탑재한 기기뿐 아니라 크롬(OS와 브라우저 둘 다)이라는 환경에서도 간단한 변환 과정을 거치면 앱을 실행할 수 있습니다. 이런 상황에서 안드로이드 앱을 실행할 수 있는 환경이 점점 증가할 것이라 예상합니다. 한편으로는 안드로이드 앱을 통해 여러 환경에서 동작하는 서비스를 만들 수 있으니 그 중요도가 높아졌으며, 다른 한편으로는 중요도와 더불어 경쟁이 매우 심해졌다고 할 수 있습니다. 이 책이 현재 개발하시는 앱이나 서비스 중인 앱의 스레드 사용에 도움이 돼서 UX를 개선하는 데 도움이 되었으면 좋겠습니다.

그리고 이 책을 쓸 수 있게 제안해 주신 김병희 님, 이 책을 세상에 내어 주신 한빛미디어의 김창수 팀장님, 정지연 님께 감사의 말씀을 드립니다. 부모님 송근태, 유재옥 님께 늘 감사합니다. 마지막으로 이 책을 쓰는 데 물심양면으로 도와준 아내 박말순 그리고 두 아들인 재익, 재빈에게 감사합니다.

대상 독자 및 참고사항

이 책은 안드로이드 앱에서 사용하는 스레드를 효율적으로 사용할 수 있게 도와줍니다. 안드로이드 앱을 개발하는 개발자라면 누구나 쉽게 읽을 수 있습니다.

이 책의 예제 코드는 다음에서 받으실 수 있습니다.

- https://github.com/mcsong/book

예제 코드를 실행하는 환경은 다음과 같습니다.

- 자바 7, 이클립스 4.4, 안드로이드 4.0.3(15)

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 찾기도 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 First – 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 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 좀 더 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 위해 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 어려운 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

chapter 1

소개

안드로이드는 리눅스 커널을 기반으로 모바일 기기에 최적화한 운영체제다. 안드로이드 애플리케이션은 자바로 개발하고, '달빅Dalvik'[01] 이라는 가상머신(Virtual Machine)을 실행환경으로 사용한다. 이 가상머신은 자바로 개발한 애플리케이션을 안드로이드에서 실행할 수 있도록 자바가상머신(Java Virtual Machine)과 같은 역할을 수행한다.

안드로이드 웹 사이트[02]에는 9억 개의 기기가 안드로이드를 사용하고 있고, 플레이 스토어[03]에는 1,000,000개 이상의 애플리케이션이 등록되어 있다. 이렇게 많은 애플리케이션이 마켓에서 사용자의 선택을 기다리는 상황에서, 애플리케이션이 마켓에서 좋은 평가를 받기 위해서는 무엇을 고려해야 할까? 애플리케이션이 사용자에게 제공하는 가치(Value)와 더불어 애플리케이션의 UI/UX가 매우 중요할 것이다. 모바일 기기는 사용자와 밀접하게 연결되어 있기 때문이다.

모바일 애플리케이션 관점에서 UI/UX에 대해서 간단하게 정의해 보자. UI(User Interface)는 인간과 애플리케이션의 접점으로 화상, 문자, 소리나 애플리케이션을 조작하는 수단으로, 대표적으로 GUI(Graphical User Interface)가 있다. UX(User Experience)는 사람이 UI를 사용해서 애플리케이션과 의사소통하는 경험이라 할 수 있다.

---

01 http://ko.wikipedia.org/wiki/달빅_(소프트웨어)

02 http://www.android.com

03 http://play.google.com

현재 많은 모바일 기기가 멀티 코어를 사용한다. 이미 멀티 코어를 사용하는 윈도, 맥 그리고 리눅스 등의 데스크톱 기반 GUI 애플리케이션에서는 UX의 반응성을 개선하기 위해서 다중 스레드를 사용하고 있다. 다중 스레드란 시스템 자원을 최대한 활용하기 위한 수단으로, 많은 작업을 동시에 수행할 수 있게 하는 것이다. 다중 스레드는 GUI 애플리케이션에서 느린 작업을 처리하느라 GUI 화면이 멈추는 문제 등을 해결하는 수단으로 사용하고 있다.

안드로이드 또한 UI 반응성을 높이기 위해서 다중 스레드 사용을 권고하고 있고 API 수준에서 다중 스레드 프로그래밍을 편리하게 할 수 있도록 지원하고 있다. 안드로이드 API는 아파치 하모니 프로젝트[04]를 기반으로 자바 5.0의 코어의 상당 부분, 아파치 HTTPClient 4.0.X 버전, JSON, XMLPullParser와 화면에 보이는 각종 위젯을 지원하고 있다. 자바 5.0에서 스레드 관련 패키지는 코어에 존재하고 있으므로 자바가 지원하는 스레드 형태 전부를 사용할 수 있다.

이 책에서는 안드로이드 애플리케이션의 UX를 개선하기 위한 여러 가지 방안 중에 반응성을 높이는 방안을 살펴보려고 한다. 그리고 개별 방안에 대한 내용은 예제 소스를 첨부하였다. 장별로 주요 내용을 언급하자면, 우선 2장에서는 안드로이드에 대한 배경지식으로 애플리케이션을 패키징하고 설치하는 과정을 살펴보고, 3장에서는 애플리케이션 GUI 위젯이 화면에 어떻게 그려지고 갱신되는지와 이 과정에서 발생할 수 있는 문제에 대해서 살펴보겠다. 4장에서 안드로이드 프로세스와 스레드, 스레드의 스케줄링에 대해서 살펴본 다음, 5장에서 안드로이드에서 사용할 수 있는 기존의 자바에서 지원하는 스레드를 살펴볼 것이다. 6장에서는 안드로이드에서 사용하는 스레드 형태를 살펴보고, 7장에서는 안드로이드 스레드 중에 AsyncTask의 소스에 대한 설명과 버전별 개선점을 확인해 본다. 8장에서는 AsyncTask를 개선해서 반응성을 높이는 방안에 대해서 살펴보고, 마지막으

04 http://en.wikipedia.org/wiki/Apache_Harmony

로 9장에서는 이 라이브러리를 이용한 애플리케이션을 개발해 보겠다.

이 책의 예제는 자바 7, 이클립스 4.4 그리고 안드로이드 4.0.3(15)을 기준으로 하고 있다.

NOTE

덧붙여, 도움이 될 만한 사이트를 몇 군데 소개한다.

1. 안드로이드 웹 사이트의 개발자 사이트[05]

안드로이드 애플리케이션을 개발하는 데 필요한 정보, 예제 그리고 각종 팁을 알려 주고 있어서 애플리케이션 개발에 매우 도움이 된다.

2. 디자인 사이트[06]

안드로이드 애플리케이션 UI 디자인에 대해서 설명하고 있다. UI 디자인과 함께 UX에 대한 내용도 설명하고 있어서 안드로이드 애플리케이션의 UI/UX를 설계하기 위해서 참고하기 매우 좋다.

3. 안드로이드 배포 사이트[07]

플레이 스토어에 애플리케이션을 런칭하기 위해서 필요한 정보, 마케팅 그리고 이미지 리소스를 제공하고 있다. 이 사이트를 사용해서 개발한 애플리케이션을 플레이 스토어에 배포할 수 있다.

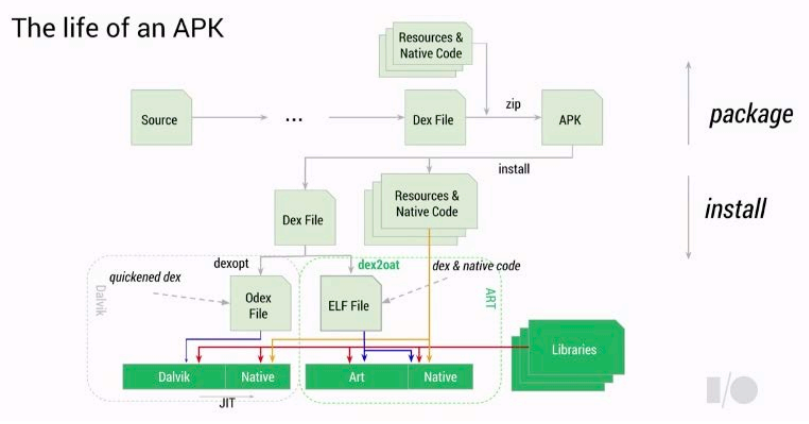
chapter 2

애플리케이션 패키징과 설치 과정

안드로이드는 리눅스 커널을 기반으로 한 모바일 운영체제로, 자바로 개발하며 달빅 가상머신을 실행환경으로 사용한다. 그러나 달빅은 실행 시간 및 성능에서 단점을 가지고 있었다. 이러한 달빅의 단점을 개선한 새로운 실행환경이 킷캣(4.4) 버전에서 등장했다. 바로 아트(Art, Android RunTime)다. 이 실행환경은 킷캣 버전의 개발자 모드에서 선택적으로 사용할 수 있다.

여기에서는 애플리케이션이 패키징되는 과정과 달빅과 아트 실행환경에서 설치되는 과정에 대해서 살펴보자.

그림 2-1 안드로이드 애플리케이션의 패키징/설치 과정



[그림 2-1]은 2014년 구글 I/O에서 발표한 자료의 일부다. 그림의 상단 부분이 애플리케이션을 패키징하는 과정이고, 하단의 과정이 패키징된 애플리케이션을

설치하는 과정이다. 설치 과정은 실행환경(달빅, 아트)에 따라 구분되는 것을 알 수 있다.

2.1 패키징 과정

안드로이드 애플리케이션 패키징이란 프로젝트를 빌드해서 .apk 파일을 만들어 내는 것을 말한다. 패키징의 결과로 .apk 파일이 만들어지고, 이 파일을 사인해서[01] 안드로이드 기기에 설치할 수 있는 애플리케이션을 만들게 된다. 이 과정은 다음과 같다.

1. 소스를 컴파일한다. 이 과정은 .java 소스 파일을 .class의 바이트코드로 변환하는 것이다.
2. 바이트코드를 dex(Dalvik EXecutables) 포맷으로 변환한다. 변환 결과로 .dex파일이 만들어진다.
3. dex 파일과 네이티브 라이브러리 그리고 리소스 등의 파일을 zip으로 압축해서 .apk 파일을 생성한다.

이 과정으로 패키징을 하게 된다. 위의 1번 과정에서는 자바 SDK에 있는 javac 컴파일러를 사용한다. 2번 과정에서는 안드로이드 SDK에 있는 dx 툴(SDK/build-tools/xx.x.x/dx.bat)을 사용해서 다음과 같이 변환한다.

[화면 2-1] dx 툴을 사용해서 .class 파일을 .dex 파일로 묶는 예제

```
C:\Temp>C:\dev\android-sdk\build-tools\17.0.0\dx.bat --dex --verbose --output
c:\Temp\sjava-android-ex\bin\classes_dx.dex c:\Temp\sjava-android-ex\bin\classes

processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\
AbstractActivity.class...
```

01 IDE에 연결해서 실행하면 기기에서 실행할 수 있다. 이 과정에서도 디버그 모드의 사인 과정을 거치게 된다.

```
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\BitmapHandler.
class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\
BookApplication.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\BuildConfig.
class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch0201Activity$1.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch0201Activity$2$1.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch0201Activity$2.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch0201Activity.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch02Activity$1.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch02Activity$2.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch02Activity$3$1.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch02Activity$3.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch02Activity$4$1.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch02Activity$4.class...
processing c:\Temp\sjava-android-ex\bin\classes\.\net\sjava\book\ch02\
Ch02Activity$5.class...
```

3번 과정도 안드로이드 SDK에 있는 aapt[Android Asset Packaging Tool] 툴(SDK/build-tools/xx.x.x/aapt.bat)을 사용한다. 이 툴을 사용해서 사인하지 않은 .apk 파일을 만들 수 있다. 안드로이드 기기에는 사인이 된 .apk 파일만 설치되므로 사인된 .apk를 만들기 위해서는 다시 한 번 apkbuilder 툴을 사용할 필요가 있다.

[화면 2-2] aapt 툴을 사용해서 .apk 파일로 패키징

```
C:\dev\android-sdk\build-tools\17.0.0\aapt package -f -M C:\Temp\sjava-android-
ex\AndroidManifest.xml -S C:\Temp\sjava-android-ex\res -j C:\Temp\sjava-
```

```
android-ex\libs\*.jar -A C:\Temp\sjava-android-ex\assets -I C:\dev\android-sdk\
platforms\android-15\android.jar -F C:\Temp\sjava-android-ex\bin\aapt-example.
apk
```

[화면 2-2]는 aapt 툴을 사용해서 사인하지 않은 애플리케이션의 패키지 파일인 aapt-example.apk를 만드는 예제다.

[화면 2-3] aapt 툴을 사용해서 .apk 파일에 패키징된 파일 확인

```
C:\Temp>C:\dev\android-sdk\build-tools\17.0.0\aapt list C:\Temp\sjava-android-
ex\bin\aapt-example.apk
res/drawable/android_01.png
res/drawable/android_02.png
res/drawable/android_03.png
res/layout/activity_ch02.xml
res/layout/activity_ch02_01.xml
res/layout/activity_ch03.xml
res/layout/activity_ch03_01.xml
res/layout/activity_main.xml
res/menu/ch0301.xml
res/menu/main.xml
AndroidManifest.xml
resources.arsc
res/drawable-hdpi/ic_launcher.png
res/drawable-mdpi/ic_launcher.png
res/drawable-xhdpi/ic_launcher.png
android/support/v4/print/PrintHelper$1.class
android/support/v4/print/PrintHelper$PrintHelperKitkatImpl.class
android/support/v4/print/PrintHelperKitkat$2$1.class
```

[화면 2-3]에서는 aapt 툴을 사용해서 패키징한 파일이 포함하고 있는 각종 파일들을 확인할 수 있다. 이 절에서 예제로 사용된 툴킷들은 안드로이드 SDK에서 제공하는 개발자 툴 사이트[02]에서 확인할 수 있다.

02 http://developer.android.com/tools/help/index.html

2.2 설치 과정

애플리케이션 설치는 .apk 파일을 안드로이드 기기에서 사용할 수 있는 형태로 필요한 설정과 실행파일 그리고 리소스를 배치하는 것을 말한다. 설치 과정은 실행환경으로 달빅을 사용하는지 아트를 사용하는지에 따라 달라진다. 달빅이 애플리케이션을 실행하기 위해서 사용하는 파일(odex)은 안드로이드 /data/dalvik-cache 폴더에서 확인할 수 있고, 아트가 사용하는 파일(elf) 역시 같은 위치에 같은 이름과 확장명을 사용하고 있다. 두 실행환경은 같은 위치의 같은 파일을 실행하지만 어떤 실행환경을 사용하는지에 따라서 설치 시에 실행파일이 달라진다.

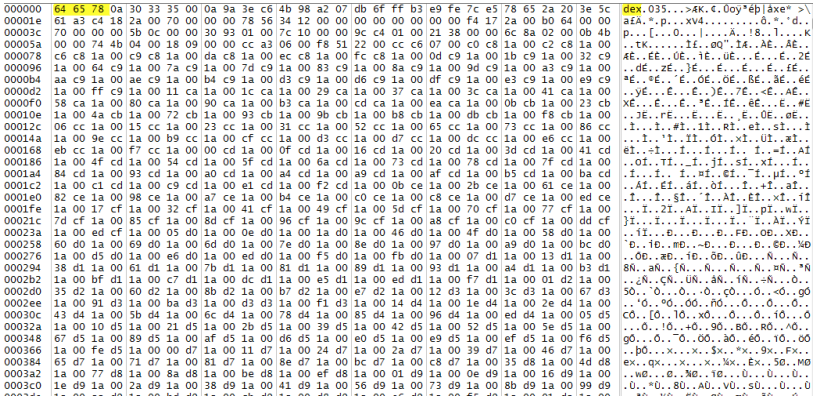
2.2.1 달빅 실행환경

달빅 실행환경을 사용하는 안드로이드 기기에 애플리케이션이 설치되는 과정은 다음과 같다.

1. 패키지 파일을 dex와 리소스 등으로 분리한다.
2. 실행파일인 dex 파일을 odex(Optimized dex) 포맷으로 변환하고, /data/dalvik-cache 폴더에 저장한다. 저장되는 odex 파일의 확장명은 dex다.

이 중에서 2번 과정은 안드로이드에 있는 dexopt 툴(/system/bin/dexopt)을 사용한다. 달빅은 애플리케이션을 실행할 때마다 바이트코드bytecode를 해석하는 과정을 거친다. 이 과정의 성능을 개선하기 위해서 안드로이드 2.2부터 JITJust-In-Time 컴파일러를 사용한다.

그림 2-2 헥사 뷰어로 확인한 dex 파일 포맷



[그림 2-2]는 달빅을 사용하는 안드로이드 기기가 사용하는 dex 파일이다. 이 파일의 헤더를 보면 dex 파일이라는 것을 알 수 있다.

2.2.2 아트 실행환경

아트 실행환경을 사용하는 안드로이드 기기에 애플리케이션이 설치되는 과정은 다음과 같다.

1. 패키지 파일을 dex와 리소스로 분리한다.
2. 실행파일인 dex 파일을 oat(elf) 포맷으로 변환하고, /data/dalvik-cache 폴더에 저장한다. 저장되는 oat 파일의 확장명은 달빅과 같이 dex를 사용한다.

이 중에서 2번 과정은 안드로이드가 내장하고 있는 dex2oat 툴(/system/bin/dex2oat)을 사용한다. 이 과정에서 dex 포맷을 머신 코드로 변환하기 위해서 AOT(Ahead-Of-Time) 컴파일러를 사용한다. 그리고 AOT 컴파일러에 의해서 변환된 oat 파일은 elf(Executable and Linking Format)[03] 형태로 데이터와 실행 명령을 포함하는 공유 객체(Shared Object)의 형태가 된다.

03 http://en.wikipedia.org/wiki/Executable_and_Linkable_Format

[화면 2-4] 루팅한 안드로이드 기기에서 oat 파일 확인 결과

```
root@toro:/ # find / -name *.oat
find / -name *.oat
./data/local/tmp/dalvik-cache/system@framework@boot.oat
./data/dalvik-cache/system@framework@boot.oat
```

[화면 2-4]는 아트를 사용하는 안드로이드 기기에서 oat 파일의 목록을 확인한 결과다. 이 화면에서 찾은 파일(/data/dalvik-cache/system@framework@boot.oat)은 시스템에서 사용되는 jar 파일 전부가 oat 파일로 변환된 결과고, 아트 실행환경을 기본 라이브러리로 사용한다.

[화면 2-5] 아트 실행환경을 사용하는 안드로이드의 애플리케이션 목록 일부

```
-rw-r--r-- system   all_a178  3420592 2014-05-07 19:16 data@app@com.microsoft.
office.officehub-1.apk@classes.dex
-rw-r--r-- system   all_a118  4796848 2014-07-16 10:02 data@app@com.microsoft.
skydrive-1.apk@classes.dex
-rw-r--r-- system   all_a94   7344560 2014-08-04 20:55 data@app@com.
mobisystems.fileman-1.apk@classes.dex.
```

[화면 2-5]는 아트를 사용하는 안드로이드 기기에 설치된 애플리케이션 목록의 일부다. 이 애플리케이션이 위치하는 폴더는 달빅의 애플리케이션이 위치하는 폴더와 동일한 /data/dalvik-cache다. 게다가 달빅이 사용하는 파일의 이름을 동일하게 사용하고 있어서 목록만 보면 달빅을 사용하는지 아트를 사용하는지 확인할 수 없다.

그림 2-3 헥사 뷰어로 확인한 oat 파일 포맷

[그림 2-3]은 아트를 사용하는 안드로이드 기기에서 /data/dalvik-cache 폴더에 있는 oat 파일을 열어본 화면이다. 이 파일의 헤더를 보면 이 파일의 포맷은 dex가 아니라 elf라는 것을 알 수 있다.

앞에서 안드로이드 애플리케이션을 패키징하고 설치하는 과정에 대해서 살펴봤다. 롤리팝부터 아트가 기본 실행환경으로 사용되면서 달빅을 대체하게 되었다. 따라서 기존에 사용하던 달빅과 이것을 대체하는 아트에 대해서 자세히 살펴볼 필요가 있다.

NOTE

달빅(Dalvik)

달빅 가상머신은 안드로이드 기기가 출시되면서부터 킷캣 버전까지 안드로이드 애플리케이션을 실행하는 환경으로 사용되었다. 달빅은 애플리케이션을 실행할 때마다 바이트코드를 읽어서 사용하므로 실행 시간이 길어지는 단점이 있었다. 이 방식의 성능 개선을 위해서 안드로이드 2.2부터 JIT[Just-In-Time] 컴파일러를 사용하였다.

JIT 컴파일러는 기존의 인터프리터 방식과 정적 컴파일 방식을 혼합해서 사용하는 방식으로, 애플리케이션을 실행하는 시점에 필요한 바이트코드를 머신 코드로 변환해서 실행하는 방식이다. JIT 컴파일러는 변환된 머신 코드를 캐싱해서, 같은 함수가 여러 번 불리는 경우 매번 기계어 코드를 생성하는 것을 방지해 주기 때문에 가상머신 기반의 환경에서 성능 개선을 위해서 많이 사용한다. JIT 컴파일러는 실행 시에 바이트코드를 읽기 때문에, 아트의 AOT 방식보다 애플리케이션을 다양한 하드웨

04 http://www.android.com/intl/ko_ALL/kitkat/

어에서 실행시킬 수 있는 장점이 있다. 하지만 애플리케이션 실행과 화면전환마다 변환 과정을 거치기 때문에 머신 코드를 사용하는 애플리케이션에 비해서 성능이 안 좋다. 그리고 빈번한 변환 과정은 배터리를 많이 소비하게 하고, 리소스(CPU, 메모리 등)를 많이 사용하는 단점이 있다.

아트(Art)

안드로이드 킷캣[04]에서 처음 등장한 아트는 애플리케이션을 설치할 때, dex2oat 툴을 사용해서 dex 포맷을 머신 코드로 변환한다. 이 dex 포맷을 머신 코드로 변환하는 데 AOT(Ahead-Of-Time) 컴파일러를 사용한다.

AOT 컴파일러는 바이트코드로 배포된 애플리케이션을 기존의 인터프리터나 JIT와 같은 방식으로 바이트코드에서 머신 코드로 변환하는 과정을 미리 수행하는 방식이다. AOT 컴파일러는 LLVM을 사용해서 애플리케이션을 ARM, ARM64, MPS, X86, X86-64 등의 다양한 CPU에서 실행할 수 있도록 지원한다.

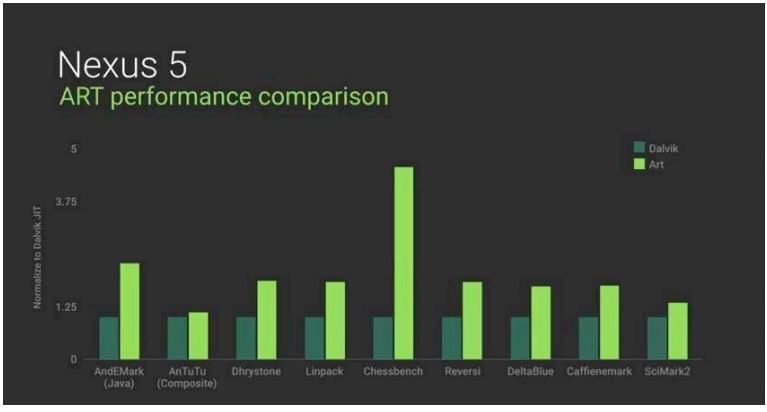
앞에서 언급한 CPU에서 64가 붙은 타입은 64bit의 CPU를 말하는데, 이를 통해 이제 안드로이드가 64비트 애플리케이션을 실행하는 환경으로 아트를 사용하리라는 것을 알 수 있다.

이상으로 달빅과 아트에 대해서 간략하게 살펴봤다. 앞에서 달빅은 애플리케이션을 실행할 때마다 바이트코드를 읽어서 실행하는 JIT 컴파일 방식을 사용하고, 아트는 미리 머신 코드로 변환해서 애플리케이션을 실행하는 AOT 컴파일 방식을 사용한다는 것을 확인했다. 따라서 달빅과 아트의 장단점은 구체적으로 애플리케이션을 실행하는 데 사용하는 컴파일 방식에 따른다고 볼 수 있다.

다시 한 번 정리하면, 달빅은 애플리케이션을 다양한 하드웨어에서 실행시킬 수 있는 장점이 있으나 애플리케이션의 성능이 안 좋고, 배터리 소비가 많으며, 리소스(CPU, 메모리 등)를 많이 사용한다는 단점이 있다. 아트는 달빅에 비해서 애플리케이션의 성능이 개선되었고, 배터리 사용시간이 증가했고, 리소스 사용량이 감소한 장점이 있다. 달빅에 비해 컴파일 과정을 한 번 더 거치기 때문에 설치 시간이 길어지고 컴파일된 파일로 설치공간이 10~20% 증가하는 단점이 있다.

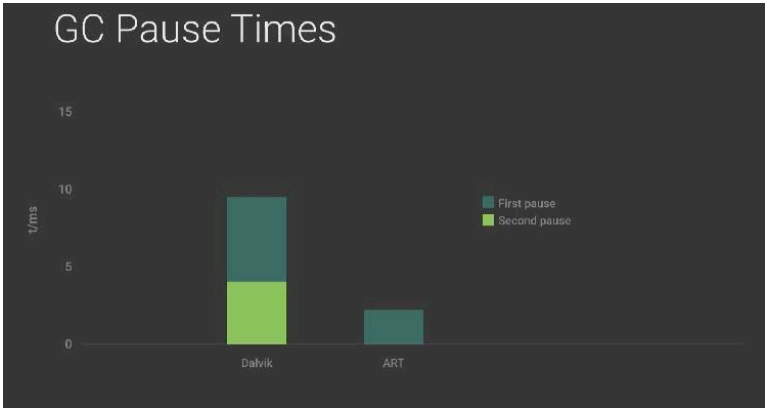
2014 구글 I/O에서 발표한 달빅과 아트의 비교 자료를 살펴보자.

그림 2-4 넥서스(Nexus) 5에서 달빅과 아트의 성능 비교



[그림 2-4]는 2014 구글 I/O 키노트에서 발표된 내용의 일부로 달빅과 아트의 성능을 비교한 자료다. 이 그림은 여러 벤치마크 애플리케이션으로 달빅과 아트의 성능을 비교했고, 아트가 달빅에 비해 평균적으로 2배 가량 성능이 좋아진 것을 확인할 수 있다.

그림 2-5 달빅과 아트의 가비지 콜렉션 중단시간 비교



[그림 2-5]도 2014 구글 I/O 키노트에서 발표된 내용의 일부로, 달빅과 아트의 가비지 콜렉션 성능을 비교한 차트다. 이 차트는 가비지 콜렉션으로 안드로이드 애플리케이션의 실행이 멈춘 시간을 보여 주고 있다. 아트를 사용하는 것이 달빅을 사용하는 것보다 가비지 컬렉션 수행으로 인한 애플리케이션 중단 시간이 짧은 것을 알 수 있다.

앞의 자료들에서 살펴본 바와 같이, 아트를 사용하는 것이 달빅을 사용하는 것보다 성능이 향상된다는 것을 알려 준다. 이제 안드로이드 OS 소스에서는 기본 실행환경으로 달빅이 삭제되고 아트를 사용하도록 변경한 소스가 안드로이드 소스 리파지토리에 커밋되었다.[05]

05 https://android-review.googlesource.com/#/c/98553/, 이 주소는 AOSP(Android Open Source Project)의 2014년 06월 18일 커밋로그이다. 로그의 내용으로 이제 더 이상 달빅 VM을 사용하지 않겠다는 것이다. 따라서, 이후부터 AOSP는 아트를 기본 런타임으로 사용하게 된다.

chapter 3

화면 그리기

시스템 앱이나 서비스만 사용하는 등의 특별한 경우를 제외하고, 대부분의 안드로이드 애플리케이션은 화면에 여러 위젯Widget을 사용해서 UI를 구성한다. 위젯은 크게 뷰View와 뷰그룹ViewGroup으로 구분할 수 있다. 뷰그룹도 뷰의 일종이지만 뷰를 담을 수 있는 컨테이너 뷰를 말한다. 안드로이드는 단일 스레드 모델Single-threaded Model을 사용하여 뷰와 뷰그룹을 그리거나 갱신한다. 한 스레드가 뷰나 뷰그룹의 요청을 차례로 실행해서 UI를 그리거나 갱신하는 구조라는 것이다.

안드로이드 애플리케이션을 실행하면 스레드가 하나 만들어지는데 이 스레드가 바로 UI 스레드다. 이 스레드는 화면을 그리고drawing, 위젯을 갱신invalidate하고, 화면을 터치하거나 키보드를 입력하는 등의 메시지를 해당 위젯에 전달하는 역할을 한다. 이 스레드가 앞에서 언급한 단일 스레드 모델의 스레드인 것이다.

예를 들어, 화면에 버튼Button 위젯이 있다고 가정해 보자. 이 버튼을 탭Tab하면, UI 스레드는 터치 이벤트를 버튼 위젯에 보내고, 버튼 위젯은 상태(pressed state)를 갱신하고 위젯의 갱신 요청(invalidate request)을 이벤트 큐에 전달한다. UI 스레드는 큐에 추가된 갱신 요청을 확인하고 버튼 위젯에게 자신을 다시 그리게drawing 알려 준다. 안드로이드 화면을 구성하는 뷰나 뷰그룹이 단일 스레드 모델로 구현되어 있기에, 뷰의 그리기나 갱신 등의 과정은 스레드 세이프Thread-Safe하지가 않다. 스레드 세이프하지 않다는 것은 다중 스레드를 사용해서 동시에 뷰에 갱신을 요청하는 경우 뷰를 그리는 과정에서 문제가 발생할 수 있다는 것이다. 그래서 안드로이드에서 UI 스레드가 아닌 일반 스레드에서 위젯의 갱신을 요청하면 예외를

발생시킨다.

UI 스레드에서 파일을 읽거나 쓰기, 네트워크로 데이터를 전송/수신하는 등의 I/O 작업을 처리하는 경우, 요청한 작업이 차례대로 처리되기 때문에 요청한 작업이 완료하기 전에는 위젯의 갱신이나 이벤트 처리 등이 블록된다. 이렇게 화면이 블록되면 안드로이드가 애플리케이션을 강제로 종료시키는 상황이 발생하는데 이런 상황을 ANR[Application Not Responding]이라고 한다.

이 장에서는 애플리케이션에서 액티비티가 화면을 구성하는 과정과 결과, 화면의 구성요소인 위젯을 그리고[drawing] 갱신하는 과정을 살펴보고, 뷰 갱신을 일반 스레드에서 요청하는 경우에 발생하는 에러 상황, ANR 그리고 에러 상황을 미리 감지할 수 있는 스트릭모드[StrictMode]에 대해서 살펴보겠다.

3.1 화면 구성

애플리케이션에서 액티비티가 실행되면 액티비티가 레이아웃을 로딩해서 UI를 그리게 된다. 이 과정에서 액티비티는 루트뷰[RootView]를 만들고, 이 뷰를 시작으로 트리[Tree] 구조로 화면에 보이는 위젯들을 구성한다. 다음의 예제를 살펴보자.

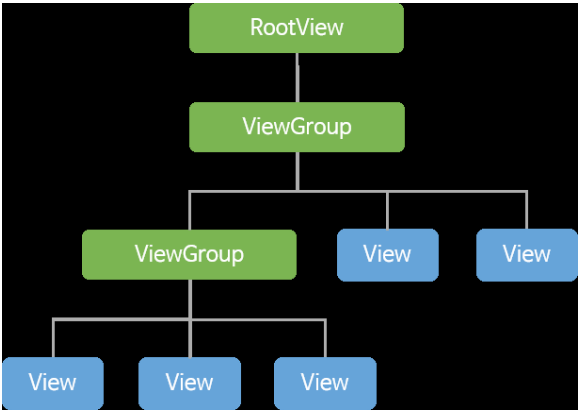
[예제 3-1] 레이아웃을 구성하는 일반적인 XML 파일

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_ch03_01_relativelayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <LinearLayout
        android:id="@+id/activity_ch03_01_linearLayout1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_marginLeft="14dp"
        android:background="@android:color/darker_gray"
```

```
        android:orientation="vertical" >

        <TextView
            android:id="@+id/activity_ch03_01_txtView_01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="InnerTextView01"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/activity_ch03_01_txtView_02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="14dp"
            android:text="InnerTextView02"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <Button
            android:id="@+id/activity_ch03_01_button_01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="InnerButton01" />
    </LinearLayout>

    <TextView
        android:id="@+id/activity_ch03_01_txtView_03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/activity_ch03_01_linearLayout1"
        android:layout_marginTop="122dp"
        android:text="TextView03"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <Button
        android:id="@+id/activity_ch03_01_button_02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/activity_ch03_01_txtView_03"
        android:layout_below="@+id/activity_ch03_01_txtView_03"
        android:layout_marginTop="40dp"
        android:text="Button02" />

</RelativeLayout>
```

[예제 3-1]은 애플리케이션 액티비티가 화면의 레이아웃을 구성하는 데 사용하는 일반적인 형태다. 이 예제의 레이아웃 트리는 다음과 같다.

그림 3-1 예제 3-1에서 XML로 구성한 레이아웃의 트리 구조



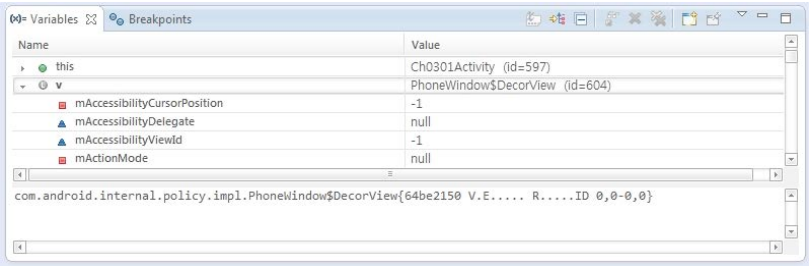
[그림 3-1]은 [예제 3-1]의 레이아웃 XML 파일을 읽어서 구성한 뷰의 트리 구조다. 애플리케이션에서 이 레이아웃 파일을 읽어서 [그림 3-1]처럼 위젯을 구성하고 있는지 확인해 보자.

[예제 3-2] 예제 3-1의 XML 레이아웃을 로딩하는 액티비티 클래스

```
public class Ch0301Activity extends AbstractActivity {
    private TextView txtView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setContentView(R.layout.activity_ch03_01);

        txtView = (TextView)findViewById(R.id.activity_ch03_01_txtView_01);

        // 루트뷰 확인을 위한 부분
        View v = txtView.getRootView();
    }
}
```

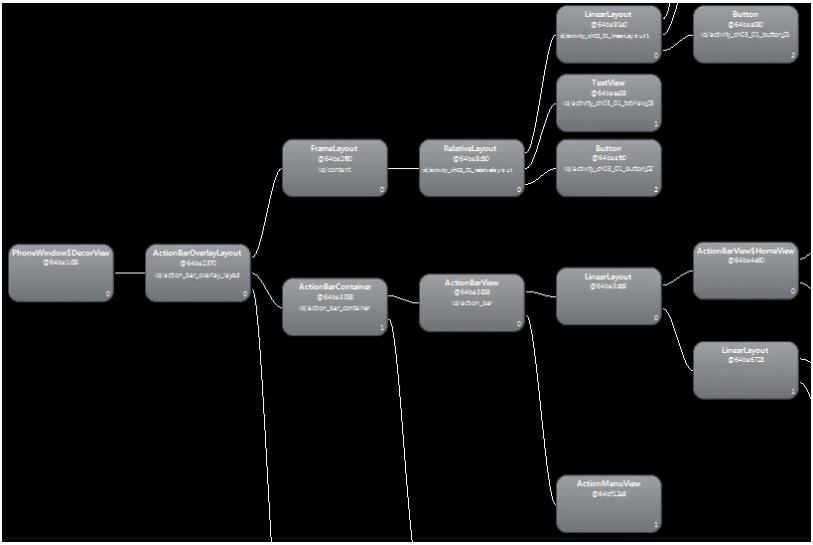
[예제 3-2]는 [예제 3-1]의 XML을 로딩하는 액티비티 클래스 생명주기에서 액티비티를 생성하는 메서드 예제다. 이 단계에서 화면을 구성하고 있는 텍스트뷰 TextView의 루트뷰를 확인할 수 있다.

그림 3-2 예제 3-2의 소스에서 루트뷰를 확인하는 디버그 창



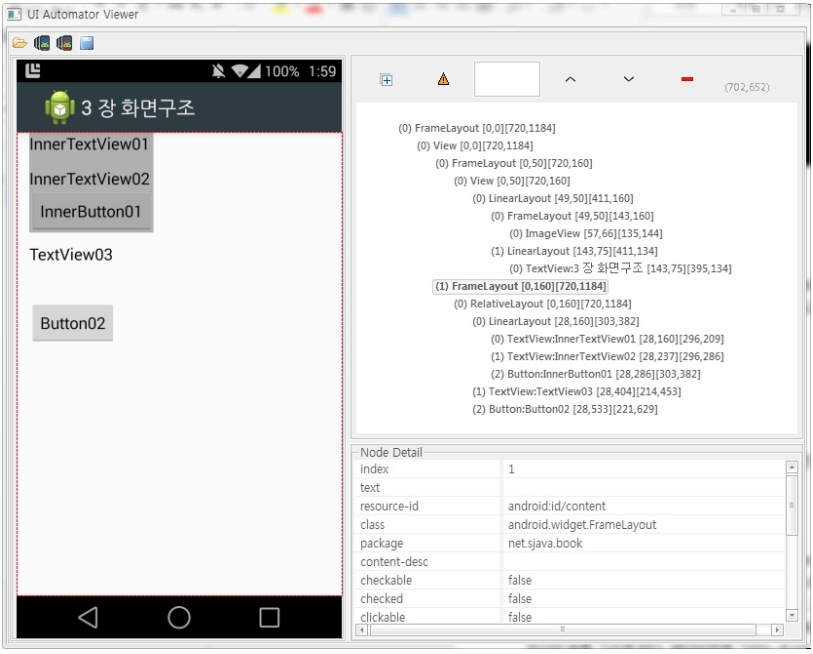
[그림 3-2]는 [예제 3-2]에서 뷰(예제에서 TextView)의 루트뷰를 확인하는 데 사용한 이클립스의 디버그 창이다. 이 화면에서 텍스트뷰의 루트뷰를 확인할 수 있다. 루트뷰 객체의 타입이 PhoneWindow$DecorView라는 것을 알 수 있다. 이 객체는 PhoneWindow 클래스(com.android.internal.policy.impl.PhoneWindow)의 중첩클래스인 DecorView 클래스의 객체라는 것도 알 수 있다. 다음으로 안드로이드 SDK가 제공하는 툴(Hierarchy Viewer, UI Automator Viewer)을 사용하여 레이아웃의 화면 구성을 확인해 보자.

그림 3-3 Hierarchy Viewer로 확인한 예제 3-2 액티비티 화면



[그림 3-3]은 안드로이드 SDK가 제공하는 툴인 Hierarchy Viewer 툴(SDK/tools/hierarchyviewer.bat)을 사용하여 확인한 액티비티 화면으로 [예제 3-2]의 액티비티가 레이아웃 파일을 읽어서 트리 구조로 화면을 구성한 것을 확인할 수 있다. 이 툴을 사용하면 액티비티가 화면을 구성하는 전체 구조(화면에 나오는 모든 부분으로 소프트웨어 버튼 영역도 포함)뿐만 아니라 [예제 3-1]의 레이아웃 파일의 위젯이 어느 부분에 위치하는 지도 확인할 수 있다.

그림 3-4 UI Automator Viewer로 확인한 예제 3-2 액티비티 화면



[그림 3-4]는 안드로이드 SDK가 제공하는 툴인 UI Automator Viewer 툴(SDK/tools/uiautomatorviewer.bat)을 사용하여 액티비티 화면을 확인한 것이다. [예제 3-2]의 액티비티가 구성한 화면을 확인할 수 있다. 이 툴로 화면을 구성하는 위젯의 트리 구조와 화면에서 위젯이 배치된 위치와 크기도 알 수 있다.

앞에서 액티비티가 레이아웃 파일을 읽어서 화면의 위젯을 트리로 구성하는 과정과 결과에 대해서 살펴봤다. 다음으로 화면을 구성하고 있는 위젯을 그리고 갱신하는 과정을 살펴보자.

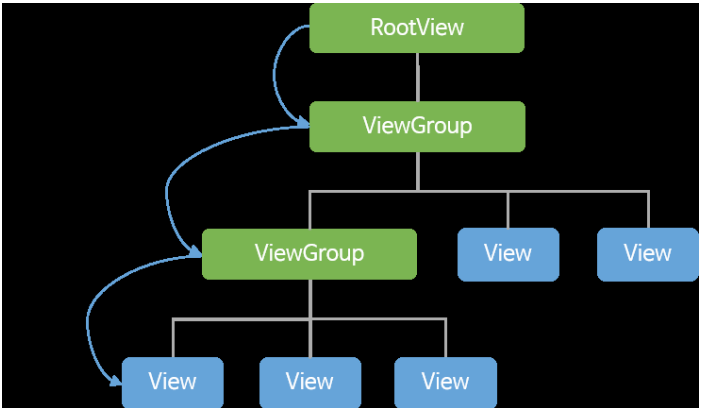
3.2 위젯 그리기와 갱신

[그림 3-1]을 보면 트리 구조의 리프 노드로 뷰가 있고, 뷰를 여러 개 가질 수 있는 뷰그룹도 있다. [그림 3-5]는 [그림 3-1]을 바탕으로 그리기(`draw()`) 과정을 표시

한 것이다. 위젯 그리기는 루트뷰에서 시작한다. 이 작업은 레이아웃 트리를 만들고 위젯의 영역을 확인하고(measure), 화면에서 만들어진 트리를 순회(traverse)하면서 그린다(draw). 그리기는 트리를 이동하면서 화면에 배치된 뷰를 차례대로 그리게 된다. 뷰그룹은 자식뷰가 그려지도록 요청(`draw()` 호출)한다. 뷰가 그리기를 요청받게 되면 자신을 그리게 된다. 트리 구조로 구성된 위젯 노드는 인오더in-order로 순회하면서 그린다. 따라서 위젯(뷰)을 그리기 위해서는 부모인 뷰그룹을 먼저 그리게 되고, 형제노드(sibling)는 좌에서 우의 순서대로 그리게 된다. 레이아웃 트리를 구성하고 난 뒤에, 화면의 위젯을 그리는 과정은 다음과 같다.

1. 레이아웃 트리를 사용해서 각 뷰의 영역을 만든다(`measure(int, int)`.
2. 레이아웃 트리를 사용해서 만들어진 영역에 뷰들을 배치한다(`layout(int, int, int, int)`).
3. 레이아웃의 위젯을 그린다(`draw()`).

그림 3-5 화면을 구성한 트리에서 그리기 과정

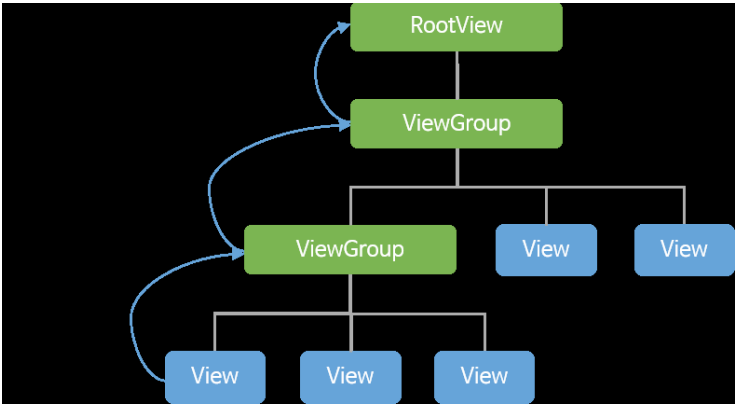


[그림 3-5]는 액티비티에서 레이아웃을 트리로 구성한 뒤에 화면을 그리는 과정을 보여 준다. 이 그림에서 보듯이 루트뷰부터 그리기를 시작한다.

화면을 갱신하는 과정은 화면 그리기를 완료한 후에 이벤트로 발생한다. 화면을

구성하는 레이아웃에 버튼 위젯이 있다고 가정하고, 이 버튼을 클릭했을 때 버튼의 색을 바꾸기 위해서 버튼을 갱신(다시 그리기)하는 과정을 살펴보자.

그림 3-6 화면을 구성한 트리의 그리기가 완료된 상태에서의 갱신 과정



[그림 3-6]은 액티비티에서 레이아웃을 트리로 구성한 뒤에, 그리기가 완료된 상태에서 화면에 있는 위젯에 이벤트가 발생하여 위젯을 갱신하는 과정을 보여 주고 있다. 이 과정은 [그림 3-6]의 `invalidate ( )` 메서드가 루트뷰에 전달되면 [그림 3-5]의 `draw ( )` 메서드를 호출해서 화면에 위젯을 다시 그리는 과정이 필요하다.

그러나 화면을 갱신하는 데 매번 이 과정을 거치는 것은 매우 비효율적이다. 그래서 허니콤에서 이 과정을 개선하기 위해서 `DisplayList`라는 것을 추가했다. `DisplayList`는 화면의 위젯을 그리기 위한 정보를 가지고 있는 목록이고, 이 목록을 사용해서 그리기가 필요한 위젯을 그리게 한다. 이 목록을 사용하기 위해서는 매니페스트 `<application>` 요소의 `android:hardwareAccelerated` 속성 값을 `true`로 설정하거나 아이스크림(14) 이상의 안드로이드를 사용해야 한다. 아이스크림 이상의 안드로이드는 이 속성의 기본값이 `true`다.

[예제 3-3] 애플리케이션의 hardwareAccelerated의 값을 설정하는 소스

```
boolean hardwareAccelerated = sa.getBoolean(
    com.android.internal.R.styleable.AndroidManifestApplication_
```

```
hardwareAccelerated,
        owner.applicationInfo.targetSdkVersion >= Build.VERSION_CODES.ICE_
CREAM_SANDWICH);
```

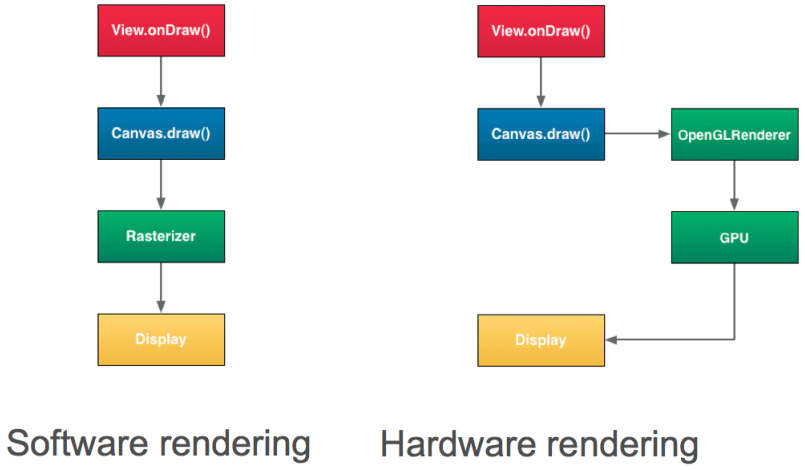
[예제 3-3]은 android.content.pm 패키지에서 확인할 수 있는 PackageParser 클래스의 소스 일부로, 이 예제는 hardwareAccelerated의 값을 설정하는 소스다. 이 소스를 보면 애플리케이션의 매니페스트 파일에 android:hardwareAccelerated의 속성값을 설정했으면 그 값을 사용한다. android:hardwareAccelerated의 속성값을 설정하지 않은 경우, android:targetSdkVersion의 값이 아이스크림(14)보다 큰 경우에는 hardwareAccelerated의 기본값으로 true를 사용하는 것을 알 수 있다.

[예제 3-4] 매니페스트 파일에 선언한 SDK 정보

```
<uses-sdk android:minSdkVersion="15" android:targetSdkVersion="19" />
```

[예제 3-4]는 예제 프로젝트가 매니페스트 파일에 선언한 SDK 버전에 대한 정보다. 안드로이드 버전별 점유율을 확인하면 85% 이상이 아이스크림 이후의 버전을 사용하고 있다. 따라서 매니페스트의 android:hardwareAccelerated 속성을 false로 설정하지 않은 애플리케이션은 안드로이드 기기의 하드웨어 가속을 기본으로 사용하게 된다.

그림 3-7 화면을 그리는 방식에 따른 과정



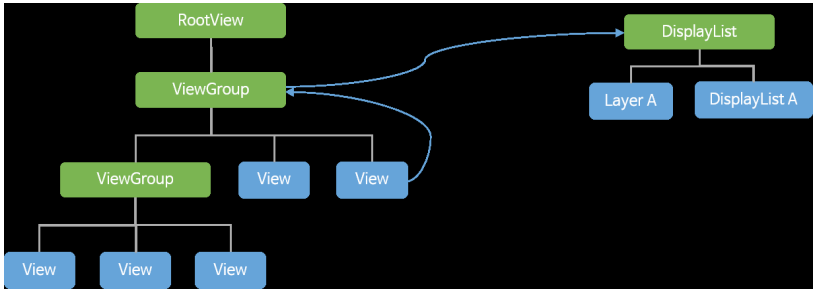
[그림 3-6]은 소프트웨어 렌더링으로 화면의 위젯을 그리는 과정과 하드웨어 가속을 사용해서 화면의 위젯을 그리는 과정이다. 소프트웨어 렌더링Software Rendering은 허니콤 이전 버전에서 사용하는 방식이고, 허니콤 이후부터는 선택적으로 하드웨어 렌더링Hardware Rendering을 사용할 수 있다. 이 두 가지 방식의 성능 차이를 살펴보자.

그림 3-8 화면을 그리는 방식에 따른 리스트뷰의 그리기 시간

Drawing a ListView			
	Hardware layer	DisplayList	Software
Time in ms	0.009	2.1	10.3

[그림 3-7]은 화면을 그리는 방식에 따른 성능비교 자료다. 하드웨어 가속을 사용한 DisplayList가 기존의 방식인 소프트웨어Software보다 5배 가량 성능이 향상된 것을 확인할 수 있다. DisplayList를 사용하는 과정을 살펴보자.

그림 3-9 DisplayList를 사용해서 화면을 갱신하는 과정



안드로이드 애플리케이션이 android:hardwareAccelerated 속성의 값을 true로 설정해서 사용하면 안드로이드 애플리케이션은 화면의 뷰를 갱신하는 과정에서 DisplayList를 사용한다. [그림 3-8]을 보면 invalidate( ) 메서드를 호출하고, 이 메서드가 호출되면 호출한 뷰를 가지고 있는 뷰그룹이 DisplayList에 뷰를 그리게 한다. 이 과정이 [그림 3-6]의 과정보다 빠르게 위젯을 갱신하므로 [그림 3-7]에서의 성능 향상을 확인할 수 있다.

3.3 위젯 갱신 에러

안드로이드 애플리케이션에서 UI 스레드가 아닌 일반 스레드로 위젯의 setXXX( ) 메서드 부류를 호출해서 위젯에 갱신을 요청하면 예외가 발생한다. 예를 들어, 일반 스레드에서 텍스트뷰 인스턴스에 Text를 변경하기 위해서 setText(" 새로운 값 ")과 같은 메서드를 호출하는 경우에는 다음과 같은 예외가 발생한다.

[화면 3-1] 일반 스레드에서 위젯을 갱신할 때에 발생하는 예외

```
W/System.err: android.view.ViewRootImpl$CalledFromWrongThreadException: Only
the original thread that created a view hierarchy can touch its views.
W/System.err:    at android.view.ViewRootImpl.checkThread(ViewRootImpl.
java:6094)
W/System.err:    at android.view.ViewRootImpl.invalidateChildInParent(ViewRoot
Impl.java:857)
```

```
W/System.err:    at android.view.ViewGroup.invalidateChild(ViewGroup.java:4320)
W/System.err:    at android.view.View.invalidate(View.java:10942)
W/System.err:    at android.view.View.invalidate(View.java:10897)
W/System.err:    at android.widget.TextView.checkForRelayout(TextView.
java:6587)
W/System.err:    at android.widget.TextView.setText(TextView.java:3813)
W/System.err:    at android.widget.TextView.setText(TextView.java:3671)
W/System.err:    at android.widget.TextView.setText(TextView.java:3646)
W/System.err:    at net.sjava.book.ch03.Ch0301Activity$InitializeThread.
run(Ch0301Activity.java:64)
```

[화면 3-1]은 일반 스레드에서 텍스트뷰의 Text를 갱신하려고 시도해서 발생한 예외의 콜 스택Call Stack이다. 이 상황을 재현하는 소스는 예제 프로젝트 3장에서 확인할 수 있다. 이 예제는 액티비티에서 일반 스레드로 텍스트뷰 위젯의 갱신을 요청했고, 위젯을 갱신하는 데 필요한 과정을 알 수 있다. 이 과정은 다음과 같다.

1. 위젯(TextView)에서 invalidate( ) 메서드를 실행한다.
2. invalidate( ) 메서드는 위젯의 상위 노드인 ViewGroup(ViewParent 타입)의 invalidateChild( ) 메서드를 호출한다.
3. invalidateChild( ) 메서드는 루트뷰에 invalidateChildInParent( ) 메서드를 호출해서 자식 노드의 뷰에게 invalidate( ) 메서드로 위젯을 다시 그리도록 한다.
4. 이때 UI 스레드가 아닌 일반 스레드에서 요청한 경우, CalledFromWrongThreadException을 발생시켜 위젯의 갱신 요청에 대해서 예외를 발생하게 한다. 이 코드는 다중 스레드를 사용하는 안드로이드에서의 화면 처리를 UI 스레드만이 할 수 있도록 해서, 일반 스레드가 위젯을 갱신하게 되면 발생하는 문제를 제거했다.

[화면 3-1]에서 예외를 발생시키는 checkThread( )의 소스를 확인해 보자.

[소스 3-1] 안드로이드 루트뷰 구현체인 ViewRootImpl 클래스의 checkThread() 소스

```
void checkThread() {
    if (mThread != Thread.currentThread()) {
        throw new CalledFromWrongThreadException(
            "Only the original thread that created a view hierarchy can touch
its views.");
    }
}
```

[소스 3-1]이 [화면 3-1]의 예외를 발생시키는 소스다. 화면에 위젯을 구성하고 그리는 스레드가 UI 스레드(mThread)인데, UI 스레드가 아닌 일반 스레드로써 위젯을 갱신할 수 없도록 제한하고 있다.

3.4 ANR

ANR[Application Not Responding]은 애플리케이션의 UI 스레드가 응답할 수 없는 상황이다. 애플리케이션이 응답하지 않는 상황이란, UI 스레드에서 파일을 읽거나 저장, 네트워크로 데이터를 전송/수신하는 등의 느린 작업을 처리하는 경우 이 작업을 처리하느라 화면이 블록되는 상황이다. 이런 느린 작업을 UI 스레드에서 처리하려고 시도하면 화면을 그리는 것이나 위젯의 이벤트가 처리되는 것이 안드로이드의 제약조건을 위배하는 경우가 발생한다. 이 경우 안드로이드가 이 애플리케이션을 강제로 종료시킨다. 안드로이드 개발자 사이트의 ANR 사이트[01]를 확인해 보면 다음의 제약조건을 확인할 수 있다.

1. 애플리케이션이 입력에 대해서 5초 이내에 반응하지 않는 경우
2. 브로드캐스트 리시버[BroadcastReceiver]가 10초 이내에 이벤트를 처리하고 결과를 반환하지 못한 경우

01 http://developer.android.com/training/articles/perf-anr.html

이 제약 조건을 위배하는 코드는 예제 Chapter03Activity.java에서 확인할 수 있다.

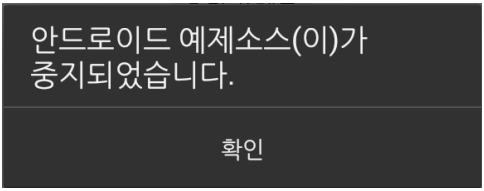
[예제 3-5] ANR 재현 코드

```
mButton01.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                File f = new File(Environment.getExternalStorageDirectory() +"/
net.sjava.book/");
                if(!f.exists())
                    f.mkdirs();

                Bitmap bitmap = null;
                BitmapHandler handler = BitmapHandler.newInstance();
                try {
                    // resource에 있는 이미지를 로딩하고 이 이미지를 UI 스레드에서 저장한다.
                    for(int i=0; i < 6; i++) {
                        bitmap = BitmapFactory.decodeResource(getResources(),
BookApplication.imageArray[i]);
                        handler.save(bitmap, BookApplication.strArray[i]);
                    }

                    // 메인 스레드이기에 화면 갱신가능
                    Log.d(TAG, Thread.currentThread().getName());
                    mTextView.setText("변경 완료");
                } catch(Throwable e) {
                    Log.e(TAG, "runOnUiThread error", e);
                }
            }
        });
    }
});
```

[예제 3-5]는 ANR을 재현하는 코드다. 맨 위에서 화면을 구성하고 있는 버튼(mButton01)에 클릭 이벤트를 등록하고 있다. 이벤트가 발생해서 처리하는 로직은 다음과 같다.

UI 스레드에서 리소스의 이미지를 이미지 개수만큼 반복해서 파일에 저장을 한다. 그리고 이미지 파일의 저장이 완료되면, 텍스트뷰(mTextView)에 완료 메시지를 갱신한다. 이 예제를 실행하고 화면을 여러 번 탭하면, 화면이 멈추는 상황이 발생하고 잠시 후에 안드로이드가 애플리케이션을 종료시키는 다이얼로그를 띄운다.

그림 3-10 안드로이드가 띄운 ANR 다이얼로그 화면



[그림 3-9]는 안드로이드가 애플리케이션을 종료한다고 띄운 다이얼로그다. 이처럼 ANR이 발생하는 상황을 피하는 방법을 몇 가지 살펴보자.

1. API에서 지원하는 비동기 클래스인 AsyncTask를 사용한다.
2. 네트워크, 데이터베이스, 파일처리, 비트맵 사이즈 변경 등의 작업처럼 느린 작업은 일반 스레드로 처리한다.
3. 기존의 Thread 클래스나 HandlerThread 클래스를 사용하기 원한다면, 스레드의 우선순위를 THREAD_PRIORITY_BACKGROUND로 변경해서 사용한다. 우선순위를 변경하기 위해서는 Process.setThreadPriority( )에 THREAD_PRIORITY_BACKGROUND 상수를 전달하면 된다. UI 스레드에서 Thread 클래스나 HandlerThread 클래스를 사용하면 이 스레드의 우선순위가 메인 스레드와 같아지기 때문이다.

안드로이드 ANR 사이트에서는 ANR이 발생하는 제약조건으로 앞에서 언급한 2가지를 제시하고 있지만 안드로이드가 ANR로 인식하는 상황은 이 2가지 상황 외에도 더 있다. 이 2가지 이외의 ANR 상황에 대해서는 ActivityManagerService 클래스가 ANR이 발생하는 타임아웃을 선언하고 있다. 이 클

래스는 com.android.server.am 패키지에서 확인할 수 있다.

[예제 3-6] 젤리빈(4.1.1)에서 ActivityManagerService 클래스에 선언된 ANR 타임아웃 값 —

```
// How long we allow a receiver to run before giving up on it.
static final int BROADCAST_FG_TIMEOUT = 10*1000;
static final int BROADCAST_BG_TIMEOUT = 60*1000;

// How long we wait for a service to finish executing.
static final int SERVICE_TIMEOUT = 20*1000;

// How long we wait until we timeout on key dispatching.
static final int KEY_DISPATCHING_TIMEOUT = 5*1000;
```

[예제 3-6]은 ActivityManagerService 클래스에서 선언된 ANR 타임아웃 값이다. ANR의 제약조건이라고 설명한 두 가지 상황에 대한 타임아웃을 확인할 수 있다. 브로드캐스트 ANR 상황에 대한 타임아웃은 10초로 선언되어 있고, 브로드캐스트를 처리하는 백그라운드 애플리케이션의 타임아웃은 60초라는 것을 알 수 있다. 키 이벤트에 대한 타임아웃은 5초로 선언되어 있다. 안드로이드의 액션에 대한 추가적인 타임아웃 정보는 앞의 클래스에서 TIMEOUT 류의 상수를 확인하면 알 수 있다. 이 상수는 ANR이 발생하는 상황에 대한 정보도 알려 주기 때문에 추가적인 ANR 상황 파악에도 도움이 된다.

3.5 스트릭모드

스트릭모드StricMode[02]는 애플리케이션의 안정성 확보 수단으로 진저브레드(9)에서 도입된 것이다. 스레드와 VM에서 발생할 수 있는 문제 상황에 대한 정책을 만들고, 애플리케이션이 이 정책을 위반하면 몇 가지 수단으로 문제를 알려 준다. 스트릭모드는 ThreadPolicy를 사용해서 UI 스레드가 파일 I/O나 네트워크 전송/

02 http://developer.android.com/reference/android/os/StrictMode.html

수신 등의 느린 작업을 시도하는 경우를 확인하고, VmPolicy를 사용해서 애플리케이션에서 발생할 수 있는 메모리 릭(데이터베이스 사용, 액티비티 등에서 발생할 수 있는 이슈)을 확인한다. 이런 상황을 스트릭모드에 등록하면 로깅, 화면의 깜박임, 애플리케이션의 종료를 통해 애플리케이션이 가지고 있는 잠재적인 문제를 알려 준다. 그래서 스트릭모드를 사용하면 애플리케이션의 안정성을 높이고 UX를 개선하는 데 도움이 된다.

허니콤(11) 이후부터는 UI 스레드에서 네트워크을 사용하여 작업하면 NetworkOnMainThreadException[03]이 발생한다. 허니콤 이전 버전에서는 네트워크 작업을 UI 스레드에서 처리해도 정책적으로 제한하지 않기 때문에 화면이 블록되는 현상을 볼 수 있고, 이 상황(ANR)에서는 안드로이드가 애플리케이션을 종료하기 때문에 안드로이드 애플리케이션이 비교적 안정적이지 않다고 느껴질 수 있다. 그래서 허니콤 이후부터는 네트워크을 사용하는 애플리케이션은 허니콤 이전 버전에 비해서 ANR이 발생할 확률이 낮아졌다. 그리고 UI 스레드에서 네트워크 작업을 할 수 없도록 강제하는 것은 애플리케이션이 허니콤 이전 버전보다 안정적으로 동작하도록 한다.

기존의 애플리케이션이 허니콤 이후의 안드로이드 기기에서 UI 스레드를 사용해서 파일, 네트워크 작업 등이 ANR이나 화면의 이벤트가 블록되는 문제를 발생시키지 않는다고 확신한다면 액티비티의 onCreate( ) 메서드에서 다음의 코드를 추가해서 스레드 정책을 제거할 수 있다. 이것으로 NetworkOnMainThreadException을 피할 수 있다.

[예제 3-7] UI 스레드에서 네트워크 작업을 할 수 있도록 하는 예제 소스

```
if(android.os.Build.VERSION.SDK_INT > 10) {
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().
permitAll().build();
```

03 http://developer.android.com/reference/android/os/NetworkOnMainThreadException.html

```
    StrictMode.setThreadPolicy(policy);
}
```

[예제 3-7]은 스레드 정책을 결정하는 코드로 모든 정책을 허용하도록 구성하고 있다. 따라서 애플리케이션의 UI 스레드에서 네트워크로 작업하는 부분이 있다고 가정하면, 이 작업을 일반 스레드로 분리하지 않고도 허니콤 이후의 안드로이드에서 정상적으로 동작하도록 한다. 하지만 네트워크로 데이터를 전송/수신하는 작업은 ANR 상황을 종종 만들수 있기에 애플리케이션 안정성에 좋지 않다. [예제 3-7]의 해결방법을 사용하는 것은 미봉책에 불과해서, 네트워크로 동작하는 코드는 반드시 일반 스레드로 분리해야 한다.

API 문서에서 StrictMode 클래스를 살펴보면, 다음과 같은 코드를 Application 클래스를 상속한 클래스나 액티비티에 추가해서 사용하도록 권장하고 있다.

[예제 3-9] 예제 프로젝트에서 사용하는 스트릭모드 예제 코드

```
if (DEVELOPER_MODE) {
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
                .detectDiskReads()
                .detectDiskWrites()
                .detectNetwork()
                .penaltyLog()
                .build());
    StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
                .detectActivityLeaks()
                .detectLeakedSqlLiteObjects()
                .detectLeakedClosableObjects()
                .penaltyLog()
                .penaltyDeath()
                .build());
}
```

[예제 3-9]는 예제 프로젝트에서 스트릭모드를 사용하는 예제 코드다. 이 예제는 개발자(디버그) 모드인 경우에만 스트릭모드를 사용해서 UI 스레드에서 네트워크

와 디스크 접근을 제한하고 메모리 릭이 발생하는 문제에 대해서 등록한다. 그리고 문제가 생기면 로그, 화면, 종료 등의 수단을 사용해서 알려 준다. 따라서 스트릭모드는 애플리케이션을 개발하는 시점에서 꼭 사용해 문제 상황을 미리 방지할 수 있는 아주 중요한 도구다.

chapter 4

안드로이드 프로세스와 스레드

안드로이드에서 애플리케이션의 실행 단위는 프로세스와 스레드다. 프로세스는 실행 중인 프로그램을 말하며, 스레드는 경량화된 프로세스라고 할 수 있다. 안드로이드 애플리케이션은 화면을 그리거나 입력을 처리하는 경우 단일 스레드를 사용하고, 파일과 네트워크 등의 작업을 처리하는 경우 다중 스레드를 사용한다. 이 다중 스레드는 한 프로세스 내에서 동작하고 UI 스레드와 일반 스레드로 구분된다. 안드로이드 애플리케이션을 실행했을 때 애플리케이션이 사용하는 컴포넌트가 하나라도 실행 중인 상태이면 컴포넌트를 실행 중인 UI 스레드를 사용해서 요청한 컴포넌트를 실행한다. 애플리케이션이 사용하는 컴포넌트 중에 실행 중인 것이 하나도 없는 상태이면 안드로이드가 리눅스 프로세스와 UI 스레드를 만들어서 요청한 컴포넌트를 실행한다. 따라서 매니페스트 파일에서 별도로 설정하지 않았다면 모든 컴포넌트는 같은 프로세스에서 UI 스레드로 실행된다.

안드로이드 애플리케이션이 사용할 수 있는 컴포넌트는 4가지로, 액티비티Activity, 서비스Service, 브로드캐스트 리시버Broadcast Receiver, 콘텐트 프로바이더Content Provider가 있다. 애플리케이션에서는 이 4가지 컴포넌트 중에 하나 이상을 매니페스트(Manifest.xml) 파일 내에서 구성하여 사용한다.