[Smali 코드 배우기]

작성: ttamna@i2sec

테스트를 위해 안드로이드 앱을 리패키징할 일이 있는데, 이 때 smali에 대한 지식이 필요하다.

그래서 웹 상에 있는 정보들을 모아 정리했다.

Understanding the dalvik bytecode with the dedexer tool.

: Dalvik 구조와, 바이트코드 등이 설명되어 있고, Smali to Java를 연습해볼수 있도록 구성되어 있음

:https://www.slideshare.net/paller/understanding-the-dalvik-bytecode-with-the-dedexer-tool

Dalvik은 **가상** 레지스터 기반으로 동작한다. 최대 64k 만큼 있을 수 있고, 대부분의 명령들은 앞의 256개의 레지스터만 사용할 수 있다.

하나의 레지스터는 하나의 값을 가지고 있을 수 있다 (char부터 float까지)

Double과 long 값은 두개의 연속된 레지스터가 필요하다.

핵심

- Dalvik 레지스터들은 지역변수처럼 동작한다.
- 메소드들은 지역변수처럼 각자의 레지스터를 갖는다.
- 호출된 메소드는 호출하는 메소드의 레지스터에 영향을 주지 않는다.

기본 타입

I - int, J - long, Z - Boolean, D - double, F - float, S - short,

C - char, V - void, [x - array (x:datatype)

메소드 종류

Static : "this" 인수가 암묵적으로 첫번째 인수로 전달되지 않으면, Static메소드.

Direct : overridde될 수 없으면 Direct메소드.

vtable 개입 없이, 직접적으로 invoke 한다.

프라이빗 메소드, 생성자

Virtual : 자식 클래스들에 의해 overridde될 수 있으면 Virtual메소드.

클래스와 관련된 vtable을 사용해 invoke 한다

인스트럭션 패밀리

```
# 레지스터간 move :
move, move/from16, move-wide, move-wide/from16, move-object, move-object/from16
```

```
# 결과값을 얻거나 세팅:
move-result, move-result-wide, move-result-object, return-void, return,
return-wide, return-object
```

```
# 예외처리:
throw, move-exception
```

```
# 레지스터에 상수 대입:
const/4, const/16, const, const/high16, const-wide/1, const-wide/32,
const-wide, const-wide/high16, const-string, const-class
```

```
# 동기화:
monitor-enter, monitor-exit
```

```
# 타입체크:
check-cast, instance-of
```

```
# 배열 조작:
new-array, array-length, filled-new-array, filled-new-array/range,
fill-array-data
```

```
# 인스턴스생성:
new-instance
```

```
# 실행 조작:
goto, goto/16, packed-switch, sparse-switch,
if-eq, if-ne, if-lt, if-ge, if-gt, if-le, if-eqz, if-nez, if-ltz, if-gez, if-gtz,
if-lez
```

```
# 비교:
cmpl-float, cmpg-float, cmpl-double, cmpl-double, cmpg-double, cmp-long
```

```
# 멤버필드에 읽기/쓰기:
iget, iget-wide, iget-object, iget-boolean, iget-byte, iget-char, iget-short,
iput, iput-wide, iput-object, iput-boolean, iput-byte, iput-char, iput-short
```

```
#배열요소에 읽기/쓰기:
aget, aget-wide, aget-object, aget-boolean, aget-byte, aget-char, aget-short,
aput, aput-wide, aput-object, aput-boolean, aput-byte, aput-char, aput-short
```

```
# 메소드 호출:
invoke-virtual, invoke-super, invoke-direct, invoke-static, invoke-interface,
invoke-virtual/range, invoke-super/range, invoke-direct/range,
invoke-static/range, invoke-interface/range
```

```
# int, long, float, double 연산 명령:
add,sub,mul,div,rem,and,or,xor,shl,shr,ushr,neg-(int, long, float, double),
not-(int, long)
```

```
# ODEX 명령:
execute-inline, invoke-direct-empty, iget-quick, iget-wide-quick,
iget-object-quick,
iput-quick, iput-wide-quick, iput-object-quick, invoke-virtual-quick,
invoke-virtual-quick/range, invoke-super-quick, invoke-super-quick/range
```

Smali to Java 연습 문제

Exercise 1

```
.method private swap([II)V
    .registers 5
    # this: v2 (Ltest10;)
    # parameter[0] : v3 ([I)
    # parameter[1] : v4 (I)
   aget           v0, v3, v4    ; v0 = v3[v4]
   add-int/lit8   v1, v4, 1     ; v1 = v4+1
   aget           v1, v3, v1    ; v1 = v3[v1]
   aput           v1, v3, v4    ; v3[v4] = v1
   add-int/lit8   v1, v4, 1     ; v1 = v4+1
   aput           v0, v3, v1    ; v3[v1] = v0
   return-void
.end method
```

Solution 1

```
private void swap( int array[], int i ){
    int temp = array[i];
    array[i] = array[i+1];
    array[i+1] = temp;
}
```

Exercise 2

```
.method private sort([I)V
       # this: v6 (Ltest10;)
       # parameter[0] : v7 ([I)
       const/4        v5, 1          # v5 = 1
       const/4        v4, 0          # v4 = 0
l2c4:  move           v0, v4         # v0 = v4
       move           v1, v4         # v1 = v4
l2c8   array-length    v2, v7         # v2 = v7.length
       sub-int/2addr   v2, v5         # v2 = v2-v5
       if-ge           v0, v2, l2ee   # if (v0 >= v2) -> l2ee
       aget            v2, v7, v0     # v2 = v7[v0]
       add-int/lit8    v3, v0, 1      # v3 = v0+1
       aget            v3, v7, v3     # v3 = v7[v3]
       if-le           v2, v3, l2e8   # if (v2 <= v3) -> l2e8
       invoke-direct   {v6, v7, v0}, Test10/swap;swap([II)V
       move            v1, v5         # v1 = v5
l2e8   add-int/lit8    v0, v0, 1      # v0 = v0+1
       goto            l2c8           # -> l2c8
l2ee   if-nez          v1, l2c4       # if (v1 != 0) -> l2c4
       return-void
```

Solution 2

```
private void sort(int arr[]) {
    boolean v1;
    do {
        v1 = false;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] > arr[i + 1]) {
                swap(arr, i);
                v1 = true;
            }
        }
    } while (v1 != false);
}
```

Exercise 3

```
const/16            v1, 8
new-array           v1, v1, [I
fill-array-data     v1, I288
invoke-direct       {v0, v1}, Test10/sort ; sort([I)V
...
I288:  data-array
      0x04, 0x00, 0x00, 0x00
      0x07, 0x00, 0x00, 0x00
      0x01, 0x00, 0x00, 0x00
      0x08, 0x00, 0x00, 0x00
      0x0A, 0x00, 0x00, 0x00
      0x02, 0x00, 0x00, 0x00
      0x01, 0x00, 0x00, 0x00
      0x05, 0x00, 0x00, 0x00
       end data-array
```

Solution 3

```
int array[] = {4, 7, 1, 8, 10, 2, 1, 5};
this.sort(array);
```

Exercise 4

```
.method private read(Ljava/io/InputStream;)I
        .registers 3
        # this: v1 (Ltest10;)
        # parameter[0] : v2 (Ljava/io/InputStream;)
        .catch java/io/IOException from :300 to :306 using :30a
:300:   invoke-virtual      {v2}, java/io/InputStream/read; read()I
:306:   move-result         v0
:308:   return              v0
:30a:   move-exception      v0
        const/4             v0, 15
        goto                :308
.end method
```

Solution 4

```
private int read(java.io.InputStream input_stream){
    int result = 0;
    try{
        result = input_stream.read();
    } catch (Exception e){
        result = 15;
    }
   return result;
}
```

example.smali

: Smali로 된 클래스를 코멘트로 설명하는데, 전반적인 내용들이 모두 포함되어 있어서 smali를 익히는데 큰 도움이 된다.

: http://androidcracking.blogspot.kr/2010/09/examplesmali.html

```
# 클래스 이름, 덤프시 파일 경로 결정
.class public Lcom/packageName/example;
# Object 에서 상속 (Activity, View 등일 수 있다)
# 클래스 구조는 다음과 같다: L<class path="">;
.super Ljava/lang/Object;
# 원래의 자바 파일 이름
.source "example.java"


# 이것들은 클래스 인스턴스 변수들이다.
.field private someString:Ljava/lang/String;

# final 변수들은 실제적으로 직접 사용되지 않는다.
# 왜냐하면 그것들에 대한 참조가 값 자체로 대체되기 때문이다
# primitive cheat sheet:
# V - void, B - byte, S - short, C - char, I - int
# J - long (두 개의 레지스터를 사용), F - float, D - double
.field public final someInt:I  # the :I integer 를 의미한다
.field public final someBool:Z # the :Z boolean 을 의미한다

# 배열을 만드는 코드   [x   (x 는 자료형)
.field public final someCharArray:[C
.field private someStringArray:[Ljava/lang/String;


# 이것은 생성자의 <init> 이다.
# 매개변수 목록은 다음과 같다: ZLjava/lang/String;I
#   Z - boolean
#   Ljava/lang/String; - 자바의 문자열 오브젝트
#       (primitive 타입이 아닌것의 뒤에는 세미콜론이 붙는다)
#   I - integer
# 그리고 이 생성자는 V 를 리턴하는데, V 는 void 를 의미한다
.method public constructor <init>(ZLjava/lang/String;I)V
    # 변수 공간이 얼만큼 필요한지 정의한다.
    # 우리는 v0, v1, v2, v3, v4 그리고 v5 를 변수로 갖는다
    # smali/baksmali 는 기본으로 .registers 를 사용한다
    # 그러나 이것은 --use-locals 옵션을 통해 바꿀 수 있다
    # apktool 은 --use-locals 과 --sequential-labels 를 사용한다
```

```
    .locals 6

    # 자바 코드에서의 Z, Ljava/lang/String, I 의 원래 이름을 알려준다
    # 이것들은 항상 존재하지는 않으며, 보통 최적화/난독화에 의해 제거된다
    .parameter "someBool"
    .parameter "someInt"
    .parameter "exampleString"

    # .prologue 와 .line 지시어는 대부분 무시할 수 있다
    # line number 는 가끔 디버깅 에러 분석에 유용하다
    .prologue
    .line 10

    # p0 은 0 번째 파라메터를 의미한다. 이 경우 p0 은 Java 클래스의 this 를 의미한다
    # 부모 클래스의 생성자를 호출한다
    # (상단에서 볼 수 있듯이) 이 경우엔 (부모 class 가) Ljava/lang/Object; 다
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    # v0 에 문자열을 저장한다
    const-string v0, "i will not fear. fear is the mind-killer."

    # v0 에 0xf 를 저장한다   (0xf 는 10 진수로 15 다)
    # 16 진수는 모든 기계 언어에서 사용한다(사람은 보통 10 진수를 사용한다)
    # 이전에 v0 에 있던 값은 사라진다
    # 변수는 그냥 레지스터고, 따로 타입은 존재하지 않는다.
    # 어떤 타입의 값이던 저장할 수 있다
    const/4 v0, 0xF

    # v1 에 StringBuilder 인스턴스를 만든다
    new-instance v1, Ljava/lang/StringBuilder;

    # StringBuilder 를 v2 로 초기화한다 (V 를 반환한다, V 는 void)
    const-string v2, "the spice must flow"
    invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/lang/String;)V

    # v1 에 p1 을 append 한다(append 메소드 호출)
    # p1 은 첫번째 파라메터다(boolean 타입의), 따라서 append(Z) 라는 코드가 된다
    # append 가 StringBuilder 를 반환하는 코드를 주목하자
    invoke-virtual {v1, p1}, Ljava/lang/StringBuilder;->append(Z)Ljava/lang/StringBuilder;

    # move-result-object 를 사용해서 위의 결과를 v1 에 저장한다
    move-result-object v1
    # object 가 아닌 것들은 move-result 또는 move-result-wide 를 사용한다

    # 우리 StringBuilder 에 v2 를 append 한다
    # append 가 boolean 타입이 아닌 String 타입을 취할 때 어떤 형태를 갖는지 주목하자
    const-string v2, "some random string"
    invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
```

```
    move-result-object v1

    # 우리 StringBuilder 의 toString 메소드를 호출한다
    invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
    move-result-object v1

    # 우리의 새 문자열을 log 로 출력한다
    # 이는 debug 에 사용할 수 있고, ddms 나 logcat 등을 통해 확인할 수 있다
    # 안드로이드 개발자 문서를 검색해 d() 함수가 뭔지 살펴보자
    const-string v0, "Tag"
    invoke-static {v0, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
    move-result v0

    # 현재 시간을 밀리초단위로 구한다
    # J 는 long (wide value, 2 개 레지스터 사용)
    invoke-static {}, Ljava/lang/System;->currentTimeMillis()J
    move-result-wide v2
    # 여기부터 v2 는 wide value 를 갖는것을 기억하자, 따라서 v2 과 v3 를 차지한다

    # 따라서 우리는 v4 를 다음 변수로 사용할 수 있다
    # 가능하면 변수를 재사용하자
    const-wide/16 v4, 0x300 # 이로인해 v4 와 v5 를 차지한다
    div-long/2addr v2, v4   # v2 를 v4 로 나눈다
    long-to-int v2, v2      # v2 를 int 형으로 바꾼다

    # Java 소스코드의 line number 와 관련된다(정확하지 않은 경우 있음)
    .line 12

    # p0(현재 인스턴스)의 someBool 변수에 p1 을 저장한다
    # 자바 코드에선 this.someBool = p1; 와 같은 형태의 코드일 것이다
    iput-boolean p1, p0, Lcom/packageName/example;->someBool:Z

    .line 14
    # this.someInt = p3
    iput p3, p0, Lcom/packageName/example;->someInt:I

    # v0 = this.someInt
    iget v0, p0, Lcom/packageName/example;->someInt:I

    # now we will invoke a static method.
    # 이제 정적 메소드를 호출할 것이다
    # {}는 파라메터가 없는것을 의미한다.
    #   Lfull-package-name;->method-name()return-value-type
    #   (풀패키지이름;->메소드이름()리턴값타입)  모든 내용이 있어야 한다
    invoke-static {}, Lcom/packageName/example;->someMethod()Ljava/lang/String;

    # invoke-virtual 과 invoke-direct 는 클래스 인스턴스 변수를 첫번째 파라메터로 취한다
```

```
    .line 16
    return-void # meditate on the void.
.end method

# 아래 메소드를 java 소스코드로 변경해보자
.method public static someMethod()Ljava/lang/String;
    # 더 적은 변수를 사용해도 될까?
    .locals 4

    new-instance v0, Ljava/lang/Long;

    invoke-static {}, Ljava/lang/System;->currentTimeMillis()J
    move-result-wide v1

    invoke-direct {v0, v1, v2}, Ljava/lang/Long;-><init>(J)V

    invoke-static {v0}, Ljava/lang/String;->valueOf(Ljava/lang/Object;)Ljava/lang/String;
    move-result-object v1

    # 그냥 return 이 아니라 return-object 명령을 사용하는 것을 주목하자
    return-object v1
.end method</class>
```

유용한 Smali snippet

```
# 디버거를 기다리도록 만드는 코드
invoke-static {}, Landroid/os/Debug;->waitForDebugger()V
```

```
# 로그 출력하도록 만드는 코드
# Log.v( “SADIEYU” , “==12306” );
#
# 레지스터를 오염시키지 않도록 주의
# 필요에 따라 레지스터를 더 사용할 수 있게 수정해도 된다

const-string v0, "SADIEYU"
const-string v1, "==12306"
invoke-static {v0, v1}, Landroid/util/Log;->v(Ljava/lang/String;Ljava/lang/String;)I
```

위 코드 조각들을 익히고 응용해서 소스코드에 삽입하면 디버깅에 큰 도움이 된다
이 외에 smali 코드가 필요할 경우 아래에서 소개하는 java2smali 플러그인을 사용하자.

유용한 안드로이드 스튜디오 플러그인들

: 아래 링크들에 접속하면 zip 파일을 받을 수 있는데, 안드로이드 스튜디오의 [File - settings] - (plugins) 에서 “Install from disk” 메뉴를 이용해서 설치할 수 있다

java2smali

: Java 코드를 Smali 코드로 바꿀 수 있다
: https://plugins.jetbrains.com/plugin/7385-java2smali

smalidea

: 안드로이드 스튜디오에서 Smali 코드를 Syntax highlighting 해준다
: https://github.com/JesusFreke/smali/wiki/smalidea