1. dalvik指令格式

**基础字节码 - 名称后缀/字节码后缀 目的寄存器 源寄存**

名称后缀是**wide**，表示数据宽度为64位

字节码后缀是**from16**，表示源寄存器为16位

move-wide/from16 vAA,vBBBB

move为**基础字节码**,即opcode。

wide为**名称后缀**,标识指令操作的数据宽度为64位。

from16为**字节码后缀**,标识源为一个16位的寄存器引用变量

vAA为**目的寄存器**, 它始终在源的前面，取值范围为v0~v255

vBBBB为**源寄存器**，取值范围为v0~v65535

Dalvik 指令集中大多数指令用到了寄存器作为目的操作数或源操

作数，其中 A/B/C/D/E/F/G/H 代表一个4位的数值， AA/BB/.../HH代表一个8位的数值，AAAA/BBBB/.../HHHH 代表一个16位的数值

**接下来我们学习13种dalvik指令的使用。**

1. **空操作指令**

空操作指令的助记符为nop。它的值为00，通常nop指令被用来

作对齐代码之用，无实际操作。

1. 数据操作指令

move

Move vA,vB 将vB寄存器的值赋给vA寄存器，源寄存器与目的寄存器都为4位。

Move/form16 vAA,vBBBB 将vBBBB寄存器的值赋给vAA寄存器，源寄存器为16位，目的寄存器为8位。

“move-wide vA, vB”：为4位的寄存器对赋值。源寄存器与目的寄存器都为4位。

“move-object vA, vB”：object是对象的意思，出现这个词即为对象。那这里就是为对象赋值。源寄存器与目的寄存器都为4位。

“move-object/from16 vAA, vBBBB”：为对象赋值。源寄存器为16位，目的寄存器为8位。

“move-object/16 vAA, vBBBB”：为对象赋值。源寄存器与目的寄存器都为16位。

“move-result vAA”：将上一个invoke类型指令操作的单字非对象结果赋给vAA寄存器。

“move-result-wide vAA”：将上一个invoke类型指令操作的双字非对象结果赋给vAA寄存器。

“move-result-object vAA"：将上一个invoke类型指令操作的对象结果赋给vAA寄存器。

“move-exception vAA”：保存运行时发生的异常到vAA寄存器，

**分析了这么多，总结起来move指令有三种作用：**

**第一种作用：进行赋值操作**

**第二种作用：move-result 接收方法返回值操作**

**第三种作用：处理异常的操作**

**3.返回指令(重点)**

**return就是返回的意思，大家根据这个英文就可以理解啦。**

“return-void”：表示函数从一个void方法返回，返回值为空。

“return vAA”：表示函数返回一个32位**非对象**类型的值，返回值寄存器为8位的寄存器vAA。

“return-wide vAA”：表示函数返回一个64位**非对象**类型的值，返

回值为8位的寄存器对vAA。

“return-object vAA”：这里面出现了object,表示函数返回一个

**对象**类型的值。返回值为8位的寄存器vAA。

**4数据定义指令(重点)**

**数据定义指令用来定义程序中用到的常量，字符串，类等数据。**

“const/4 vA, #+B”：将数值符号扩展为32位后赋给寄存器vA。

“const/16 vAA, #+BBBB”：将数据符号扩展为32位后赋给寄存器

vAA。

“const vAA, #+BBBBBBBB”：将数值赋给寄存器vAA。

“const/high16 vAA, #+BBBB0000“：将数值右边零扩展为32位后赋给

寄存器vAA。

“const-wide/16 vAA, #+BBBB”：将数值符号扩展为64位后赋给寄存

器对vAA。

“const-wide/32 vAA, #+BBBBBBBB”：将数值符号扩展为64位后赋给

寄存器vAA。

“const-wide vAA, #+BBBBBBBBBBBBBBBB”：将数值赋给寄存器vAA。

“const-wide/high16 vAA, #+BBBB000000000000”：将数值右边零扩展

为64位后赋给寄存器vAA。

“const-string vAA, string@BBBB”：

通过字符串索引构造一个字符串并赋给寄存器vAA。

“const-string/jumbo vAA, string@BBBBBBBB”：

通过字符串索引（较大）构造一个字符串并赋给寄存器vAA。

“const-class vAA, type@BBBB”：通过类型索引获取一个类引用并赋给寄存器vAA。

“const-class/jumbo vAAAA, type@BBBBBBBB”：

通过给定的类型索引获取一个类引用并赋给寄存器vAAAA。

这条指令占用两个字节，值为0xooff（Android4.0中新增的指令）。

**5.实例操作指令**

“check-cast vAA, type@BBBB”：check-cast v0 ,

将vAA寄存器中的对象引用转换成指定的类型。

如果失败会报出ClassCastException异常。

如果类型B指定的是基本类型，对于非基本类型的A来说，运行时

始终会失败。

“instance-of vA, vB, ”：

判断vB寄存器中的对象引用是否可以转换成指定的类型。

如果可以vA寄存器赋值为1，否则vA寄存器赋值为0。

“new-instance vAA, type@BBBB”：

构造一个指定类型对象的新实例，并将对象引用赋值给vAA寄存器。类型符type指定的类型不能是数组类。

**6、数组操作指令**

数组操作包括获取数组长度，新建数组，数组赋值，数组元素取值与赋值等操作。

“array-length vA, vB”：

获取给定vB寄存器中数组的长度并将值赋给vA寄存器。

数组长度指的是数组的条目个数。

“new-array vA, vB, type@CCCC”：

构造指定类型（type@CCCC）与大小（vB）的数组，并将值赋给vA寄存器。

“filled-new-array {vC, vD, vE, vF, vG},type@BBBB”：

构造指定类型（type@BBBB）与大小（vA）的数组并填充数组内容。vA寄存器是隐含使用的，除了指定数组的大小外还指定了参数的个数，vC~vG是使用到的参数寄存序列。

“filled-new-array/range {vCCCC  ..vNNNN}, type@BBBB”：

指令功能与“filled-new-array {vC,vD,vE,vF,vG},type@BBBB”相同，只是参数寄存器使用range字节码后缀指定了取值范围 ，vC是第一个参数寄存器，N = A +C -1。

“fill-array-data vAA, +BBBBBBBB”：

用指定的数据来填充数组，vAA寄存器为数组引用，引用必须为基

础类型的数组，在指令后面会紧跟一个数据表。

**7、异常指令**

“throw vAA”抛出vAA寄存器中指定类型的异常。

**8、跳转指令(重点)**

跳转指令用于从当前地址跳转到指定的偏移处。

**Dalvik指令集中有三种跳转指令**：

（1）goto：无条件跳转

（2）switch：分支跳转

packed-switch：有规律跳转

sparse-switch: 无规律跳转

（3）if：条件跳转

if-eq：等于**/**if-ne：不等于

if-lt：小于**/**if-le：小于等于

if-gt：大于**/**if-ge：大于等于

if-eqz：等于0**/**if-nez：不等于0

if-ltz：小于0**/**if-lez：小于等于0

if-gtz：大于0**/**if-gez：大于等于0

**9、比较指令(cmp)**

**比较指令用于对两个寄存器的值（浮点型或长整型）进行比较。**

**大于(1)/等于(0)/小于(-1)=>cmpg、cmp**

**大于(-1)/等于(0)/小于(1)=>cmpl**

例如：cmp-long vAA, vBB, vCC

比较两个长整型数。如果vBB寄存器大于vCC寄存器，

则结果为1，相等则结果为0，小则结果为-1。

例如：cmpl-float vAA, vBB, vCC

比较两个单精度浮点数。如果vBB寄存器大于vCC寄存器，

结果为-1，相等则结果为0，小于的话结果为1。

例如：cmpl-double vAA, vBB, vCC

比较两个单精度浮点数。如果vBB寄存器大于vCC寄存器，

结果为-1，相等则结果为0，小于的话结果为1。

例如：cmpg-float vAA, vBB, vCC

比较两个单精度浮点数。如果vBB寄存器大于vCC寄存器，

结果为-1，相等则结果为0，小于的话结果为1。

例如：cmpg-double vAA, vBB, vCC

比较两个单精度浮点数。如果vBB寄存器大于vCC寄存器，

结果为-1，相等则结果为0，小于的话结果为1。

**10、字段操作指令**

普通字段 => iget读 / iput 写

静态字段 => sget读 / sput 写

1. **方法调用指令(重点)**

**根据方法类型不同，共有5条方法调用指令**

invoke-virtual ：调用实例的虚方法

invoke-super ：调用实例的父类/基类方法

invoke-direct ：调用实例的直接方法

invoke-static ：调用实例的静态方法

invoke-interface ：调用实例的接口方法

**12、数据转换指令**

**数据转换指令用于将一种类型的数值转换成另一种类型。**

**它的格式为“opcode vA, vB”，vB寄存器存放需要转换的数据，转换后的结果保存在vA寄存器中。**

neg-数据类型 **=>** 求补

not-数据类型 **=>** 求反

数据类型1-to-数据类型2 **=>** 将数据类型1转换为数据类型2

**13、数据运算指令**

add/sub/mul/div/rem 加/减/乘/除/模

and/or/xor 与/或/异或

shl/shr/ushr 有符号左移/有符号右移/无符号右移