文章编号:1671-4598(2018)05-0112-04 DOI:10.16526/j.cnki.11—4762/tp.2018.05.028 中图分类号:TP3 文献标识码:A

Linux 程序向 Android 平台移植的研究

何兴鹏，刘钊远，陶琛嵘
(西安邮电大学 计算机学院，西安 710061)

摘要：针对 Linux 程序向 Android 平台移植的问题，从 ABI 层面分析了 Linux 和 Android 平台的差异，提出并研究了 Linux 程序 ABI 兼容的关键问题：系统目录结构一致性、程序加载和链接等问题；在此基础上，利用目录结构重定向和程序依赖关系分析等技术，设计实现了一种基于 ABI 兼容技术的移植方法；以移植 Linux 系统上的 CUPS 打印程序为例，对所提方法做出验证；实验结果表明本方法能够移植复杂的程序，且相比现有基于交叉编译的移植方法复杂度低、通用性高。

关键词：软件移植；系统调用；Linux 内核；应用二进制接口；Change Root 技术

Research of Porting Application from Linux to Android

He Xingpeng, Liu Zhaoyuan, Tao Chengrong
(School of Computer Science and Technology, Xi'an University of Posts and Telecommunications, Xi'an 710061, China)

Abstract: Aiming at the problem of porting the application from Linux to Android, the differences in Linux and Android from the ABI level were analyzed, The key issues of Linux ABI compatibility are proposed and studied, including the system directory structure consistency issue, the program loading and linking issue and other issues. On this basis, a porting method based on ABI compatible technology is implemented, which uses directory structure redirection and program dependency analysis techniques. The proposed method is verified by porting the CUPS program on Linux. The experimental result shows that this method can port complex applications, and compared with the cross compiler based transplantation method is low complexity, high versatility.

Keywords: software porting; system call; linux kernel; application binary interface; change root technology

0 引言

Android 是 Google 公司在 2007 年发布的一款基于 Linux 内核的开源操作系统。经过不断地发展，该操作系统不仅在移动设备上迅速普及[1]，在其它领域的使用率也逐渐增多，延伸到电视盒、桌面设备甚至工控机器。但在这些拓展领域上，Android 操作系统的应用生态极为贫乏。如果能够移植现有的 Linux 程序到 Android 平台，则能够加速 Android 平台在各个领域的应用。Linux 系统一般情况下指的是 GNU/Linux 操作系统，它是由 GNU 工程创建的一系列软件工具包，与 Linux 内核组合在一起形成的操作系统[2]，可安装在各种硬件设备中。Linux 系统经过长时间的发展，拥有丰富的软件资源[3]，可满足各种领域的需求。

目前 Linux 程序向 Android 平台移植主要采用基于交叉编译的方法。基于交叉编译的移植方法为嵌入式系统制作程序的常用方法，就是在一个平台上通过编译器编译另一个平台上的可执行程序的方法[4]。对于 Android 平台，最常用的交叉编译工具是 Google 开发的 Android NDK 或 GCC 交叉编译器。论文“移植 Tcpdump 到 Android 手机捕获无线数据包的方法研究”[5]中就使用 Android NDK 进行移植。由于 Android 中的 C 运行库是 Bionic，而 Linux 程序均使用 Glibc 库进行编程，Bionic 库与 Glibc 库具有较大不同，所以该方法需要对编译选项和目标程序代码进行适当的修改来适配 Android 的 Bionic 库。因此该方法只适合依赖程序库少且代码量较小的程序，对于依赖很多程序库或者代码量太大的程序使用 Android NDK 移植并不现实。GCC 交叉编译器虽然可以使用 Glibc 库，从而不用大量修改程序的代码，但是它需要程序提供静态编译选项且移植过程繁杂。以 Linux 系统中的 CUPS 程序为例，它依赖的程序库数量非常多，并且很多依赖程序甚至没有静态编译选项，以至于无法利用 GCC 交叉编译器进行静态编译。所以已有的移植方法不能满足移植 Linux 程序到 Android 平台的要求，需要一个更可行的移植方法。

针对 Linux 程序向 Android 平台移植的问题，本文从应用程序二进制接口（Application Binary Interface，ABI）层面分析了 Linux 和 Android 平台的差异，提出一种基于 ABI 兼容技术的移植方法。

1 平台的 ABI 差异分析

应用程序二进制接口描述了应用程序和操作系统之间，或一个应用和它的依赖库之间，或应用的组成部分之间的低层接口[6]。ABI 包含各种细节：如系统调用约定、可执行程序的格式、程序依赖库、数据类型的规定以及系统目录结构等内容。在 ABI 兼容的系统中，编译好的程序可以直接运行，无需改动目标程序代码且无需重新编译。比较 ABI 兼容性的前提是操作系统可以运行在相同处理器架构的硬件兼容环境中，否则无法比较。比如 ARM 架构下的 Linux 系统可以与 ARM 架构下的 Android 系统比较 ABI 兼容性。由于 Linux 系统与 Android 系统都能在主流处理器架构下运行，所以不存在某个处理器架构下的 Android 系统没有对应 Linux 系统的情况。因此本文的讨论不限于某个处理器架构，并且只需要在操作系统层面讨论 Linux 和 Android 操作系统的 ABI 差异。

1.1 系统调用

Linux 和 Android 可分为用户态和内核态两个部分，从该

收稿日期:2017-08-13； **修回日期**:2017-10-18。

作者简介:何兴鹏(1993-)，男，西安邮电大学，硕士研究生，主要从事嵌入式系统研究与开发方向的研究。

角度看 Linux 和 Android 的系统架构，如图 1 所示。应用程序运行在用户态，系统调用是内核态与用户态交互的接口。在内核态中，两种系统使用的都是 Linux 内核。但 Android 中的 Linux 内核与标准 Linux 内核不完全相同，其保留原有信号等机制的同时添加了 Binder IPC、Ashmem 等机制[7]。对比 Android 的 Linux 内核分支源码和标准 Linux 内核源码，发现 X86、ARM 等构架对应的系统调用均完全相同，则说明 Android 的 Linux 内核与标准 Linux 内核的系统调用保持一致。因此在相同处理器架构设备上的 Linux 系统或者 Android 系统，其系统调用的约定是一致的。比如：X86 上的 Linux 程序想在 X86 上的 Android 系统中运行，系统调用部分是兼容的。

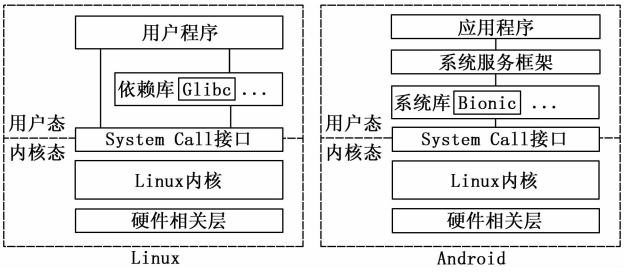

图 1　Linux 和 Android 系统架构

1.2　系统目录结构

由于 Linux 系统和 Android 系统都使用了 Linux 内核，所以内核态中的 ABI 基本相同。在用户态，有一个重要 ABI 差异就是系统目录结构。

Linux 中的系统目录结构遵循文件系统层次化标准（Filesystem Hierarchy Standard，FHS）[8]。FHS 标准规定了系统不同目录里存放的内容，如：/var 目录存放系统运行时要改变的数据、/etc 目录用来存放程序配置文件。但是 Android 没有遵循 FHS 标准，Android 中没有/var 目录、应用程序没有/etc 目录的读写权限。因此系统目录结构不一致会导致程序运行时出现无法创建或打开文件的错误。

1.3　可执行程序的格式

Linux 系统中的可执行程序可以分为脚本程序和 ELF（Executable and Linking Format）程序两种[9]。

1）脚本程序即用 Shell 等脚本语言编写的程序。脚本程序通常被其它程序调用，来辅助完成其它程序的功能。Linux 上的脚本程序遵循 POSIX 标准。POSIX 表示可移植操作系统接口，是 UNIX 系统的一个设计标准，其中包括对 Shell 脚本的规定。对于这类程序，需要使用脚本解释器执行，比如：Bash 解释器可以用来解释执行 Shell 脚本。脚本解释器文件本身则属于第二种 ELF 程序类别。因此 Linux 程序在 Android 平台运行需要解决的是 ELF 程序的执行。

2）Linux 系统里的 ELF 程序主要是指利用 C/C++等语言编写的程序，这类程序需要经过编译链接，生成 ELF 格式文件才能执行。ELF 文件有三类：可执行文件、共享库文件和重定位文件[9]。其中的可执行文件和共享库文件是程序运行时需要的文件，ABI 兼容需要解决这两种文件的执行问题。ELF 可执行程序在编译链接时有动态链接和静态链接两种方式。静态链接的程序将依赖函数与程序本身打包在一起，而动态链接的程序需要在启动的时候读取指定路径的共享库文件才可以运行。

在 Android 应用层，采用的是 Android 应用程序包（Android application package，APK）格式，它在安装到系统后由 ART 虚拟机解释执行，而 Android 底层仍然使用 ELF 可执行文件，且 ELF 文件格式和 Linux 系统中的相同。由于 Android 中的动态链接库解释器与 Linux 系统中的解释器不同，并且缺少对应的动态链接库，实际上 Linux 中的大部分 ELF 程序并不能在 Android 系统直接执行。

1.4　程序依赖库

程序依赖库指的是一系列为了简化编程复杂度、提高开发效率，对系统调用进行包装的函数集合。程序依赖库在 Linux 中编译后就是后缀为 so 的 ELF 共享库文件。ELF 共享库文件也被称为动态链接库，因为动态链接的 ELF 可执行文件在启动的时候需要加载链接其依赖的共享库文件。例如：图 1 中的 Glibc 和 Bionic 库。Linux 中程序的 C 运行库是 Glibc，而 Android 中的 C 运行库则是 Bionic。基于 Glibc 库开发的动态链接程序要想在 Android 平台上运行，如果不修改源代码适配 Bionic 库，就必须把 Glibc 共享库文件也移植过去。

ELF 动态链接程序的文件结构如图 2 所示，包含 ELF 头（ELF header）、程序头表（Program header table）、中间若干段 section 和节区头表（Section header table）。其中，.dynamic 段记录了该 ELF 程序动态链接信息。该段里的 DT _ NEED 类型的项记录了其依赖的共享库文件的位置，动态链接库解释器通过该项字段便能找到其依赖的全部动态链接库。

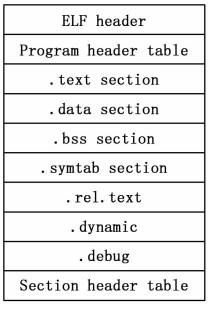

图 2　ELF 动态链接程序文件结构

2　ABI 兼容的关键技术分析

通过前面的分析可知，Linux 和 Android 平台程序的 ABI 兼容问题主要是系统目录结构一致性问题和可执行程序的加载和链接问题。这两个问题分别可以通过目录结构重定向技术和程序依赖关系分析技术解决。

2.1　目录结构重定向

为了解决系统目录结构的差异，可使用目录结构重定向技术在 Android 上构造 Linux 中的系统目录结构。目录结构重定向指改变程序执行时包括根目录在内的目录文件位置，或使得程序只看见指定的目录文件。在 Linux 系统里有两个机制能够重定向目录结构，Namespace 机制和 Change Root 机制。

Namespace 是 Linux 内核里的一种机制，提供一种包括目录在内的资源隔离功能，但是由于很多 Android 设备厂商在编译内核时没有开启 namespace 机制，所以使用 Namespace 机制不能保证对所有 Android 设备可用。

Change Root 机制指改变程序执行根目录的机制。Change Root 机制最常见的实现是 Chroot，它是 Unix 类系统中的一个操作命令。在 Linux 系统中，它的功能由 Linux 内核实现，通

过系统调用的形式提供，并且使用者需要拥有 Root 权限。Chroot 工具通常用来限制程序的可读写文件，保护系统安全。如“在 Solaris 环境下通过实现 Chroot 增进系统安全性”论文中的应用[10]，正是利用 Chroot 的特性创建一个封闭的环境，在里面提供程序所需的所有文件。

但是 Chroot 有个限制，它需要 Root 权限才能够使用。相比之下，可以使用一个名为 Proot 的工具[11]，能够在用户态实现 Chroot 的功能。它的原理与 Chroot 不同，它通过 Ptrace 系统调用，实现父进程控制子进程的执行过程，从而改变运行时的根目录。Proot 的出现拓宽了 Change Root 机制的应用场景。尤其在 Android 系统中，应用程序通常无法获取 Root 权限，这时可以使用 Proot 工具代替 Chroot 完成目录结构重定向的功能。

2.2 程序依赖关系分析

为了使 Linux 动态编译的程序能够在 Android 平台上运行，首先需要移植 Linux 中的动态链接库解释器“ld－linux.so”，Linux 动态编译的程序便能通过它在 Android 平台上加载进入内存。其次，动态链接库解释器会链接其依赖的全部动态链接库文件。本文使用程序依赖关系分析技术来解决程序的依赖文件问题。

这里给出依赖关系的数学描述：设有关系模式 $R(U)$，其中 $U\{A_1,A_2,\ldots,A_n\}$ 是关系的属性全集，X、Y 是 U 的属性子集，设 t 和 u 是关系 R 上的任意两个元组，如果 t 和 u 在 X 的投影 $t[X]=u[X]$ 推出 $t[Y]=u[Y]$，即：$t[X]=u[X]=>t[Y]=u[Y]$，则称 X 决定 Y，或 Y 依赖于 X。记为 $X->Y$。

在 Linux 中，ELF 动态链接程序之间的依赖关系表现为，关系模式 $R(U)$，其中 $U\{A1,A2,\ldots,A_n\}$ 是系统全部的 ELF 动态链接程序，X、Y 是 U 中任意两个 ELF 动态链接程序子集，如果在某个条件 t 下总能从 $t[X]=u[X]$ 推出 $t[Y]=u[Y]$，则说明两个动态链接程序具有依赖关系。这样的依赖关系在 ELF 程序间普遍存在，依赖关系的实现依靠 ELF 动态链接程序文件里的动态链接信息和软链接文件。

从图 2 中已经了解到 ELF 动态链接文件和共享库的依赖信息记录在 .dynamic 段里。在 Linux 系统中有一个 ldd 命令，它能够调用程序动态链接库解释器查看该文件 .dynamic 段里的动态链接信息。借助 ldd 命令便能分析出程序的依赖关系。但存在 ELF 文件里的动态链接信息所指向的目标是一个软链接，软链接经过若干次链接，指向最终的动态链接文件。因此程序依赖关系分析的流程如图 3 所示。

图 3 中的步骤描述如下：

1）以目标程序编译后的输出文件夹为起点，扫描里面所有的 ELF 动态链接文件；

2）读取. dynamic 段里的动态链接信息，依次查找里面记录的依赖文件。

3）将查找到的文件复制到指定位置，如果该文件是 ELF 文件，则读取该文件里的动态链接信息，进入递归；如果是软链接文件，则找到最终指向的 ELF 文件，然后递归读取。

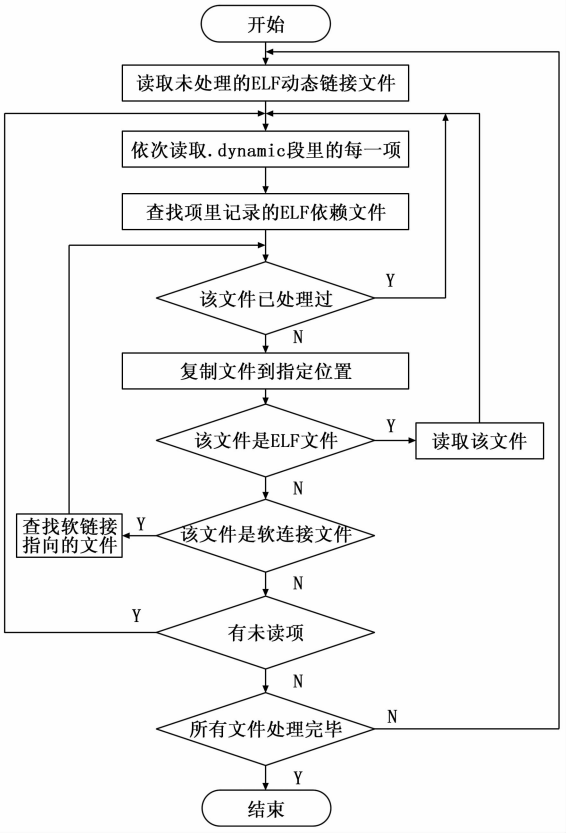

图 3 程序依赖关系分析流程图

3 基于 ABI 兼容技术的移植方法

3.1 移植方法的实现

基于 ABI 兼容的移植方法框架如图 4 所示。整体结构分为左右两个部分，左边是 Linux 目标程序部分，即需要移植的程序，右边是 Android 控制程序部分，Android 控制程序用于操作目标程序。

最底层是 Android 系统的 Linux 内核，移植的程序需要与 Android 系统共享一个内核。Linux 目标程序通过 ABI 兼容层越过 Android Framework 层直接与 Linux 内核进行系统调用操作。

中间的 ABI 兼容层由三个子模块组成，分别是 Change Root 模块 、Bash 环境模块和程序依赖模块。Change Root 模块通过 Proot 工具重定向目录结构，在一个封闭的环境里提供与 Linux 系统一致的目录结构；Bash 环境模块使用 Busybox 工具集[12]提供满足 POSIX 标准的 Shell 环境，提供程序所需的 Shell 命令；程序依赖模块通过程序依赖关系分析技术提供 Linux 目标程序所需的全部动态依赖库。三个模块共同组成的 ABI 兼容层提供了 Linux 目标程序完整的执行环境，使得 Linux 目标程序能够在 Android 平台正常运行。

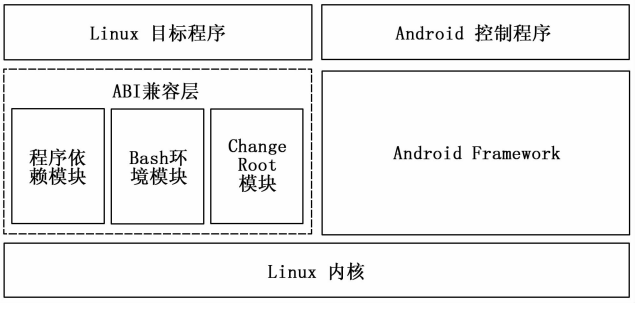

图 4 移植方法的框架

在 Linux 系统中程序原有的控制方式是命令行，但是在 Android 系统中以这种用户不友好的方式运行程序不合适，所以本文提出的方法通过编写一个 Android 应用程序，实现图形界面控制到目标程序命令控制转换的功能，使用者就能够通过 Android 控制程序这个中间层实现对目标程序的控制操作。

图 5 是目标程序的运行流程。使用者操作 Android 控制程序，由 Android 控制程序通过命令行方式直接向内核发送 fork 系统调用，从而启动 Linux 目标程序。然后 Linux 目标程序运行在 Change Root 模块所创建的进程中，并且能够从程序依赖模块获取所依赖的共享库文件。以及在需要的时候通过 Bash 环境模块运行 Shell 命令，从而完成目标程序的全部功能。

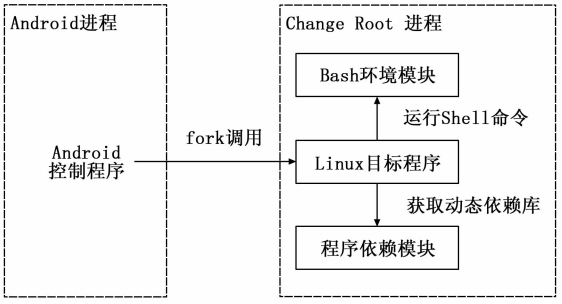

图 5　目标程序的运行流程

3.2　CUPS 程序移植测试

CUPS 是 Unix/Linux 下的开源打印系统，提供一整套完善的打印方案，拥有 USB 打印、网络打印等功能。本文以移植 Linux 中的 CUPS 程序到 OPENTHOS 系统为例，验证本文提出的移植方法。OPENTHOS 系统是清华大学、同方股份有限公司、一铭软件股份有限公司联合开发以 Android－x86 为基础的开源桌面操作系统。移植 CUPS 程序到 OPENTHOS 系统是整个 OPENTHOS 项目的一个重要部分，为该系统提供打印功能。

在 OPENTHOS 中运行移植的 CUPS 程序，并通过 ps 命令查询系统中的进程。部分查询结果如图 6 所示，其中名为 cupsd 的条目代表 CUPS 程序的进程，说明 CUPS 程序正在系统中运行。用 USB 接口连接上 HP P1108 打印机，并运行安装在 OPENTHOS 系统上的文本处理程序 WPS，使用其中的打印功能。移植的程序能够正常识别 HP P1108 打印机，并且成功打印出 WPS 里的文本，打印结果与预览中的效果一致，说明 CUPS 打印程序在 OPENTHOS 系统上运行正常。

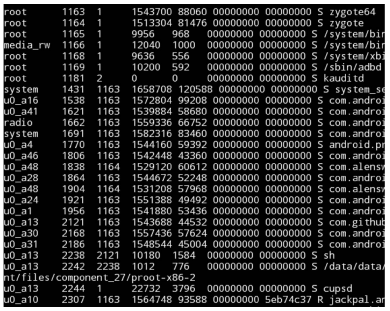

图 6　进程列表截图

3.3　移植方法对比

通过整理实验数据得到本文方法与基于交叉编译的移植方法的对比表格 1。从中可以看到，本文提出的基于 ABI 兼容的移植方法，相比基于交叉编译的方法，由于需要移植程序依赖的动态库，所以占用空间稍微大了一些，但是本文提出的方法不需要修改目标程序的代码，因此支持更多的程序、降低了移植的复杂度。由于 ABI 兼容不需要重新编译程序，所以在相同处理器构架下的设备中能够使用闭源程序。但是，本文提出的方法还有一些不足，比如：若目标程序没有提供命令行操作接口，那么这个程序很难使用本文的方法移植，但是这样的程序在 Linux 系统中很少见。另外在不同的处理器构架下闭源驱动不能使用，这个问题以后可以考虑加入 X86 等处理器架构的模拟器来运行闭源部分的程序，比如名为 Bochs 的开源 X86 平台模拟器可以在 ARM 上运行 X86 程序。

表 1　移植方法的对比

	基于交叉编译的方法	基于 ABI 兼容的移植方法
适用的程序	小型程序	绝大部分程序
闭源程序	不支持	同构架下支持
修改代码	需要	不需要
移植复杂度	复杂	简单

4　结论

本文首先分析了 Linux 和 Android 系统在 ABI 兼容上的主要差异，然后分析了消除 ABI 差异的技术：目录结构重定向和程序依赖关系分析技术。在此基础上，本文提出基于 ABI 兼容技术的移植方法，并将 Linux 系统中的 CUPS 程序成功移植到基于 Android 的 OPENTHOS 系统。本方法适用于一般的 Linux 平台的程序，并优于基于交叉编译的移植方法。希望本文提出的改进方法能够对需要把 Liunx 程序向 Android 平台移植的工作者提供一点帮助和启示。

参考文献：

[1] 贡知洲，路昭亮. Android 发展的分析与研究 [J]. 价值工程，2013，32 (2)：185－186.

[2] Mackinnon J G. The Linux operating system：Debian GNU/Linux [J]. Journal of Applied Econometrics，1999，14 (4)：443－52.

[3] 李其锋，李　兵. 开源软件开发者的演化研究 [J]. 计算机科学，2015，42 (12)：43－46.

[4] 张欢庆，高　丽，宋承祥. 基于 ARM 的嵌入式 Linux 交叉编译环境的研究与实现 [J]. 计算机与数字工程，2012，40 (2)：151－153.

[5] 胡锦晖，胡大斌. 移植 Tcpdump 到 Android 手机捕获无线数据包的方法研究 [J]. 无线通信技术，2015，24 (1)：6－10.

[6] 黄聪会，陈　靖，罗　樵，等. 面向二进制移植的虚拟化技术 [J]. 计算机应用研究，2012，29 (11)：4185－4188.

[7] 梁　超. Android 内核与标准 Linux 内核对比分析 [J]. 工业设计，2012 (2)：74－74.

[8] Hierarchy F S. Linux Standard Base [Z]. 2015.

[9] 黄　进. Linux 应用二进制兼容技术的研究与实现 [D]. 哈尔滨：国防科学技术大学，2003.

[10] 苏　雅，王　钢，巴特尔. 在 Solaris 环境下通过实现 chroot 增进系统安全性 [J]. 内蒙古工业大学学报：自然科学版，2003，22 (4)：289－292.

[11] cedric － vincent，Proot 项目主页 [EB/OL]. http：//proot.me，2017.

[12] 邵长彬，李洪亮. 用 Busybox 制作嵌入式 Linux 根文件系统 [J]. 微计算机信息，2007，23 (29)：48－50.