Android 中的微型云

Android 是 Linux® 操作系统的一个特殊发行版，目的在于管理最智能的手机中提供的功能。对于一名程序员来说，Linux 中的一切内容 — 设备、电子表格或最喜爱的一首歌曲 — 都以文件的形式出现。也就是说，可以使用一组通用的方法打开、提取、操作或保存这些不同的抽象实体的信息内容。而这种通用性形成了 UNIX® 的核心理念：一切皆为文件。

常用缩略词

- **MIME**：多用途 Internet 邮件扩展
- **HTML**：超文本标记语言
- **SDK**：软件开发工具箱
- **WWW**：万维网

文件被组织为一个有用的层次结构，称为文件系统。一个 Android 手机通常使用两种文件系统：使用非挥发（non-volatile）内存，或者使用插入式 SD 卡。两者都表现为根目录下的子目录（/）。 因此，一种使您能够在浏览器中方便地浏览这些文件系统的工具是很有用的。这个项目定义了一个使用本机语言 C 语言编写的小程序，使您能够从工作站或直接从手机内置浏览器中浏览 Android 手机的文件系统。它提供了具有超链接的页面，使您能够在层次结构树中上下移动。通过单击列出的各种文件类型，您可以查看每一个项。

设置开发环境

首先，确保您拥有一个 “获得根授权的” Android 手机，以便您能够执行类似运行 Terminal 并执行 `su` 命令这样的操作来获取根权限。如果您不知道如何对 Android 手机执行 “根授权” 操作并获取这些权限，那么快速搜索一下 Internet 应当能找到答案。（注意这一过程有时也称为越狱（*jail-breaking*）。参见 参考资料 获得有关此内容的链接）。

Android 社区主要使用两种 SDK。最常见的一种是高级 Android SDK，它使您能够用 Java™ 语言编写代码，并常使用 Eclipse 编写、测试和调试代码。另一种 SDK 较少见，即 Android 内核源代码，被保存在名为 *git* 的存储库中。（要获得这个存储库的链接，请参见 参考资料）。

由于本文主要关注创建通常位于手机的 system/bin 目录中的低级 Web 服务器，因此必须下载并安装完整的 Android 内核源代码和用于构建它的 GNU 工具。Android Kernel Source 项目的主页（参见 参考资料）提供了使用 *repo* 脚本下载整个平台的指导说明。

这个微型云软件将被交叉编译到您的工作站上的 ARM 平台中，因此确保安装了所有必要的工具。使用 `apt-get` 安装这些工具，如清单 1 所示。

清单 **1.** 安装交叉编译工具

```
$ sudo apt-get install git-core gnupg \
sun-java5-jdk flex bison gperf libsdl-dev \
libesd0-dev libwxgtk2.6-dev build-essential \
zip curl libncurses5-dev zlib1g-dev
```

回页首

开发目录的结构

使用目录名 mydroid 安装 Android 内核源代码。该名称生成了一个有效的根目录，因此回到您的主目录并发出 `mkdir mydroid` 命令。然后使用 `cd mydroid` 并在其中发出 `repo sync` 命令。

当 `repo sync` 命令将所有 Android 源代码都下载到您的 mydriod 目录后，将在其中创建许多子目录，包括：

- 您将在 mydroid/external 中创建一个 cloud 目录。该项目的源代码（cloud.c）将放在这个目录中。
- 当运行 make 命令构建好 Android 系统后，您将在 out/target/product/generic/system/bin 目录内部发现二进制文件 cloud。

回页首

Android cloud 项目

当 cloud 程序启动后，它将立即检查是否有任何命令行参数被发送给它。两个可选的预期参数分别为要进行监视的端口和要从中启动的主目录。如果这两个参数都没有指定，那么程序将默认使用标准端口 80 和默认的主目录，默认目录即为程序启动时的当前工作目录。

在完成启动后，程序将初始化 TCP/IP 套接字以通过前面提到的端口“侦听”对它的调用，然后将自身转换为一个守护进程，等待浏览器调用并提供服务。当浏览器调用这个微型云服务器的默认“页面”时，代码通过返回前面提到的“主”目录的目录清单来做出回应。其中列出了包含超链接和未包含超链接的文件名，这取决于它们是已知的文件类型还是目录。如果是已知的文件类型，这表明该文件具 有一个对应的 MIME 类型（在 WWW 世界中）。例如，在 Android 手机的内核，音频铃声被存储在 .ogg 文件中。MIME 类型 audio/ogg 告诉浏览器用扬声器播放该文件，假设您的浏览器在此方面已经进行了正确配置。

另一个出现在文件清单顶部的超链接是 Parent Directory。单击该链接将允许您向上浏览文件系统层次结构，直到您到达最高的一级：根目录。在此过程中，其他子目录名均显示为一个超链接；如果单击它们，您将进入到该子目录，查看其中包含的文件。

因此，这个 cloud 应用程序非常有用，它可以方便地浏览手机文件系统，并且源代码（参见 下载）为您提供了一个良好的功能模板，可在其基础上根据需要修改。本文结束部分给出了修改代码的建议方法。（您也可以编译代码并在自己的工作站上运行，使您也能够浏览工作站的文件系统）。有关这些内容的具体步骤，请参见 参考资料）。

回页首

C 源代码

每个程序的源代码都应当尽可能少地显示程序名称和作者。清单 2 显示了归属、include 文件（您将在其中使用定义）以及一些有用的常量。

清单 2. 归属、include 文件和有用的常量

```
// Android Cloud Application by Bill Zimmerly.
// Based on "NWEB" by Nigel Griffiths.

#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define TRUE      -1
#define FALSE     0
#define SBUF      1048576
```

```
#define LBUF      4096
#define LAPN      64
#define ERROR     1
#define LOG       2
#define LOGGING   FALSE
```

清单 3 提供了全局工作存储，可由程序中定义的所有函数访问。当浏览器调用微型云时，大多数缓冲区指针被分配了存储。当将内容返回给浏览器时，所分配的内容被返回给手机。对于资源有限的计算机来说，这项实践要比其他许多考虑事项更加重要。

清单 **3.** 全局工作存储

```
char* about="</pre>Cloud is a simple application that enables "
      "web-based browsing of the Android file system "
      "and is intended for use by people who like to "
      "explore the lowest levels of their device. It "
      "runs on a rooted Android phone and is only for "
      "browsing the file system. See the IBM "
      "developerWorks article for a full description "
      "of this application.<pre>";

char* mainbuf;
char* theDir;
char* thePort[8];
char* theList;
char* fstr;

char logDir[LBUF];

int ret;
```

在这个程序中，要正确处理文件的内容，浏览器需要知道文件扩展名与 MIME 类型之间的一个简单的映射表。清单 4 中所示的结构由 mimeokay 函数使用，用来实现以下用途：将全局字符串指针 fstr 设置为与文件扩展名 fext 相匹配的合适的 MIME 类型。通过这一信息，微型云可以显示文件名并伴随一个超链接，或者显示为不包含链接的普通文本。它还有一个作用，那就是在将文件内容发送给浏览器时提供 MIME 类型。

清单 **4.** 判断 **MIME** 类型

```
struct
{
 char *ext;
 char *mimetype;
}

mimes [] =
{
 {".htm",  "text/html"  },
 {".html", "text/html"  },
 {".xml",  "text/xml"   },
 {".gif",  "image/gif"  },
 {".jpg",  "image/jpeg" },
 {".jpeg", "image/jpeg" },
 {".png",  "image/png"  },
 {".log",  "text/plain" },
```

```
 {".conf", "text/plain" },
 {".rc",   "text/plain" },
 {".sh",   "text/plain" },
 {".prop", "text/plain" },
 {".txt",  "text/plain" },
 {".TXT",  "text/plain" },
 {".cpp",  "text/plain" },
 {".c",    "text/plain" },
 {".h",    "text/plain" },
 {".ogg",  "audio/ogg" },
 {0,0}
};

void mimeokay(char* fext)
{
 int buflen;
 int len;
 long i;

 buflen=strlen(fext);
 fstr = (char*) 0;

 for(i=0; mimes[i].ext != 0; i++)
 {
  len = strlen(mimes[i].ext);

  if(!strncmp(&fext[buflen-len], mimes[i].ext, len))
  {
   fstr=mimes[i].mimetype;
   break;
  }
 }
}
```

清单 5 显示了一部分可选代码，可以在开发和测试期间将 Android 云行为记录到一个 cloud.log 文件中。由于云运行在相对较慢的硬件上，因此最好禁用这一功能，除非您在修改代码时确实需要了解代码中发生的行为。如果希望启用这个日志记录功能，那么 清单 1 中的 LOGGING 常量应当为 TRUE，但是如果是在手机上运行的话，应当将其设为 FALSE，因为性能损失是非常明显的。

清单 **5.** 记录行为

```
void aclog(int type, char *s1, char *s2, int num)
{
 int fd ;
 char aclogbuffer[LBUF];
 char logFile[LBUF];

 if(!LOGGING)
  return;

 switch(type)
 {
  case ERROR:
   sprintf(aclogbuffer, "ERROR: %s:%s Error Number=%d, PID=%d",
        s1, s2,
        errno, getpid());
   break;
  case LOG:
   sprintf(aclogbuffer, "INFO: %s:%s:%d", s1, s2, num);
```

```
    break;
  }

  strcpy(logFile, logDir);
  strcat(logFile, "/cloud.log");

  if((fd = open(logFile, O_CREAT | O_WRONLY | O_APPEND,
         0644)) >= 0)
  {
   ret=write(fd, aclogbuffer, strlen(aclogbuffer));
   ret=write(fd, "\n", 1);
   close(fd);
  }

  if(type == ERROR)
   exit(3);
 }
```

清单 6 提供了一些为创建微型云的 HTML 布局而定义的函数。注意，具有良好设计的 Web 站点的每个页面都应当维护一个一致的外观，而 buildbuf 函数就是定义整体外观的位置。

清单 **6.** 生成 **HTML** 输出

```
 void apname(void)
 {
  strcat(mainbuf, "Android Cloud Application");
 }

 void buildbuf(char* data)
 {
  mainbuf[0]=0;
  strcat(mainbuf, "<html>\n<head>\n<title>");
  apname();
  strcat(mainbuf, "</title>\n</head>\n");
  strcat(mainbuf, "<body>\n<h3>");
  apname();
  strcat(mainbuf, "</h3>\n");
  strcat(mainbuf, "<a href=\"/About_\">About</a><br>\n");
  strcat(mainbuf, "<a href=\"/Home_\">Home</a><br>\n");
  strcat(mainbuf, "<hr>\n");
  strcat(mainbuf, "Dir: ");
  strcat(mainbuf, theDir);
  strcat(mainbuf, "<br>\n<hr>\n<pre>\n");
  strcat(mainbuf, data);
  strcat(mainbuf, "\n</pre>\n");
  strcat(mainbuf, "</body>\n</html>\n");
 }

 void htmlout(int fd, char* data)
 {
  fstr=mimes[0].mimetype;
  sprintf(mainbuf, "HTTP/1.0 200 OK\r\nContent-Type: %s\r\n\r\n", fstr);
  ret=write(fd, mainbuf, strlen(mainbuf));
  buildbuf(data);
  ret=write(fd, mainbuf, strlen(mainbuf));
 }

 void error404(int fd)
 {
```

```
 fstr=mimes[0].mimetype;
 sprintf(mainbuf, "HTTP/1.0 404 OK\r\nContent-Type: %s\r\n\r\n", fstr);
 ret=write(fd, mainbuf, strlen(mainbuf));
 buildbuf("404 Error - File not found!");
 ret=write(fd, mainbuf, strlen(mainbuf));
}
```

清单 7 展示了微型云如何返回具有指定 MIME 类型的文件内容。注意，浏览器如何要求将从服务器发送的 MIME 类型作为 Content-Type: 字符串。只需要单击由云生成的链接，就将调用该函数来返回文件的内容。然而，如果在浏览器的地址栏编辑了一个错误的文件名，那么 error404 函数（在前面定义）将向用户通知这一错误。

清单 **7.** 返回文件内容

```
void retfile(int fd, int hit)
{
 int file_fd;
 long ret;

 mimeokay(mainbuf);

 if(fstr == 0)
 {
  error404(fd);
  return;
 }

 if((file_fd = open(&mainbuf[4], O_RDONLY)) == -1)
 {
  error404(fd);
 }
 else
 {
  aclog(LOG, "SEND", &mainbuf[4], hit);

  sprintf(mainbuf, "HTTP/1.0 200 OK\r\nContent-Type: %s\r\n\r\n", fstr);
  ret=write(fd, mainbuf, strlen(mainbuf));

  while((ret=read(file_fd, mainbuf, SBUF)) > 0 )
  {
   ret=write(fd, mainbuf, ret);
  }
 }
}
```

清单 8 展示了微型云如何构建包含超链接的主文件清单。hyper 函数被使用了多次，因此将其构建为一个单独的功能是明智的。isDir 值将完全路径前面的 /CD_ 文本放在文件名的开始处，这样微型云就知道它必须显示该目录的内容。fileList 函数实际上是这个应用程序的真正的核心。

清单 **8.** 目录清单功能

```
void hyper(int isDir, char* name)
{
 strcat(theList, "<a href=\"");
```

```
 if(isDir)
  strcat(theList, "/CD_");

 strcat(theList, (char*) theDir);

 if(strcmp(theDir, "/"))
  strcat(theList, "/");

 strcat(theList, name);
 strcat(theList, "\">");
 strcat(theList, name);
 strcat(theList, "</a>");
 strcat(theList, "\n");
}

char* fileList(void)
{
 struct dirent **namelist;
 int n;
 long i;
 long j;

 theList[0]=0;

 n=scandir(".", &namelist, 0, (void*) alphasort);

 if (n < 0)
  perror("scandir");
 else
 {
  for(i=0; i<n; i++)
  {
   if(namelist[i]->d_type == DT_DIR)
   {
    if(!strcmp(namelist[i]->d_name, "."))
    {
     // strcat(theList, namelist[i]->d_name);
    }
    else if(!strcmp(namelist[i]->d_name, ".."))
    {
     if(strcmp(theDir, "/"))
     {
      strcat(theList, "<a href=\"");
      strcat(theList, "/CD_");
      strcat(theList, (char*) theDir);

      j=strlen(theList);

      while(j--)
      {
       if(theList[j] == '/')
       {
        theList[j]=0;
        j=1;
       }
      }

      if(!strcmp(&theList[strlen(theList)-4], "/CD_"))
      {
       strcat(theList, "/");
      }
```

```
       strcat(theList, "\">Parent Directory</a>");
       strcat(theList, "\n");
      }
     }
     else
      hyper(TRUE, namelist[i]->d_name);
    }
    else
    {
     mimeokay(namelist[i]->d_name);

     if(fstr == 0)
     {
      strcat(theList, namelist[i]->d_name);
      strcat(theList, "\n");
     }
     else
      hyper(FALSE, namelist[i]->d_name);
    }

    free(namelist[i]);
   }

   free(namelist);
  }

  return theList;
 }
```

在清单 9 中，定义了微型云服务器的 child 功能。这个功能在每次服务器接收到浏览器请求时运行，并且它提供了一个简单功能：分配必需的缓冲以满足请求、处理缓冲，然后释放缓冲，这样，在不需要使用 内存的时候，分配的系统内存就会降低。在手机中，内存是一种非常稀缺的资源，因此，当程序处理完一个请求后，应当将所占用的内存返回给系统。

清单 9. Daemon child 功能

```
 void child(int fd, int hit)
 {
  long i;
  long ret;
  char* cret;

  mainbuf=malloc(SBUF+1);
  theList=malloc(SBUF+1);
  theDir=malloc(LBUF+1);
  cret=getcwd(theDir, LBUF);

  ret=read(fd, mainbuf, SBUF);

  if(ret == 0 || ret == -1)
  {
   error404(fd);
  }
  else
  {
   if(ret > 0 && ret < SBUF)
    mainbuf[ret]=0;
   else
    mainbuf[0]=0;
```

```
  for(i=0; i<ret; i++)
   if(mainbuf[i] == '\r' || mainbuf[i] == '\n')
    mainbuf[i]='*';

  aclog(LOG, "request", mainbuf, hit);

  for(i=4; i < SBUF; i++)
  {
   if(mainbuf[i] == ' ')
   {
    mainbuf[i]=0;
    break;
   }
  }

  if(!strncmp(&mainbuf[0], "GET /\0", 6) ||
    !strncmp(&mainbuf[0], "get /\0", 6))
  {
   htmlout(fd, fileList());
  }
  else
  {
   if(!strncmp(&mainbuf[5], "About_", 6))
   {
    htmlout(fd, about);
   }
   else if(!strncmp(&mainbuf[5], "Home_", 5))
   {
    htmlout(fd, fileList());
   }
   else if(!strncmp(&mainbuf[5], "CD_", 3))
   {
    if(chdir(&mainbuf[8]) == -1)
    {
     error404(fd);
    }
    else
    {
     if(strcmp(theDir, &mainbuf[8]))
      strcpy(theDir, &mainbuf[8]);

     htmlout(fd, fileList());
    }
   }
   else
   {
    retfile(fd, hit);
   }
  }
 }

 free(theDir);
 free(theList);
 free(mainbuf);
 sleep(1);
 exit(1);
}
```

tiny clound 的 main（parent）功能在清单 10 中定义。它分配了将在其上侦听浏览器请求调用

的 TCP/IP 套接字。随后它将初始化一些全局变量，比如 theDir，微型云将在这些全局变量中启动。最终，它将自身创建为一个驻留程序（也称为 *daemon*），这样就可以在运行其他进程的同时在后台安静地处理浏览器请求。

清单 **10. Main** 函数

```
  int main(int argc, char **argv)
  {
   char* str;
   char* cret;

   static struct sockaddr_in cli_addr;
   static struct sockaddr_in serv_addr;

   socklen_t length;

   int i;
   int port;
   int pid;
   int listenfd;
   int socketfd;
   int hit;

   cret=getcwd(logDir, LBUF);

   if(argc < 2)
   {
    strcpy((char*) thePort, "80");
    port=atoi((char*) thePort);
   }
   else
   {
    if(!strcmp(argv[1], "-?"))
    {
     printf("Usage: cloud [Port Directory]\n");
     exit(0);
    }
    strcpy((char*) thePort, argv[1]);
    port=atoi((char*) thePort);

    if(port < 0 || port > 60000)
     aclog(ERROR, "Invalid port number (try 1 --> 60000)", argv[1], 0);

    if(chdir(argv[2]) == -1)
    {
     printf("ERROR: Invalid directory %s\n", argv[2]);
     exit(4);
    }
   }

   if(fork() != 0)
    return 0;

   signal(SIGCHLD, SIG_IGN);
   signal(SIGHUP, SIG_IGN);

   for(i=0; i<32; i++)
    close(i);

   setpgrp();
```

```
 aclog(LOG, "Cloud Port/PID=", (char*) thePort, getpid());

 if((listenfd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
  aclog(ERROR, "syscall", "socket", 0);

 serv_addr.sin_family      = AF_INET;
 serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
 serv_addr.sin_port        = htons(port);

 if(bind(listenfd, (struct sockaddr*) &serv_addr,
     sizeof(serv_addr)) < 0)
  aclog(ERROR, "syscall", "bind", 0);

 if(listen(listenfd, 64) <0)
  aclog(ERROR, "syscall", "listen", 0);

 for(hit=1; ;hit++)
 {
  length=sizeof(cli_addr);

  if((socketfd=accept(listenfd,
            (struct sockaddr*) &cli_addr,
            (socklen_t*) &length)) < 0)
   aclog(ERROR, "syscall", "accept", 0);

  if((pid=fork()) < 0)
  {
   aclog(ERROR, "syscall", "fork", 0);
  }
  else
  {
   if(pid == 0)
   {
    close(listenfd);
    child(socketfd, hit);
   }
   else
   {
    close(socketfd);
   }
  }
 }
}
```

回页首

编译、部署和测试应用程序

您需要另外创建一个文件来使用 kernel 内核源 makefile 文件恰当地编译这个云应用程序。创建一个名为 Android.mk 的文件，然后将清单 11 中的内容粘贴到该文件中。（该文件也包含在 下载 部分的源代码中）。

清单 11. Android.mk

```
ifneq ($(TARGET_SIMULATOR),true)

LOCAL_PATH:= $(call my-dir)
```

```
include $(CLEAR_VARS)
LOCAL_SRC_FILES:= cloud.c
LOCAL_MODULE := cloud
LOCAL_STATIC_LIBRARIES := libcutils libc
include $(BUILD_EXECUTABLE)

endif  # TARGET_SIMULATOR != true
```

导航到 Android 内核源代码的外部目录，创建一个名为 cloud 的子目录，然后将 cloud.c 和 Android.mk 文件全部放入该目录中。您现在可以开始构建一个新的 Android 内核系统了，微型云应用程序就将位于这个系统中。

移动到 Android 内核源代码的根目录，输入 make，然后等待一会。这一过程会花些时间，因此放轻松些，让系统完成它的工作。

如果一切顺利并且系统完成了编译，您应当会在 out/target/product/generic/system/bin 目录中找到微型云 二进制文件。您无需将整个发行版安装到您的已获得根权限的 Android 手机中：只需要将云二进制文件复制到 SD 卡。清单 12 展示了如何在您的主机上完成这一过程（假设 mydroid 目录包含所有 Android 内核源代码）。

清单 **12.** 编译内核和云

```
$ cd mydroid/external
$ mkdir cloud
$ cd cloud
$ cp ~/src/tinycloud/cloud.c .
$ cp ~/src/tinycloud/Android.mk .
$ cd ../..
$ make

--- Android system "make" messages scroll up the screen for a long time. ---

$ cd out/target/product/generic/system/bin
$ cp cloud /media/ANDROIDSDCARD/.
```

注意，/media/ANDROIDSDCARD 认为您已经将手机连接到计算机并且已经进行了设置。并且，SD 卡的名称可能是不同的。查看 /media 子目录（如果在 Ubuntu Linux 下运行的话）获得正确的名称。

运行 Android Market 免费提供的 Terminal 程序将允许您在 Android 手机中运行一个 shell 会话。Android 云一般驻留在 system/bin 目录，但这不是必须的。为了满足测试需求，可以将它放在不同的目录中。在 /data 下创建一个目录并将其命名为 *cloud*。将 /sdcard 目录中的 cloud 二进制文件复制到 /data/cloud。然后运行 chmod 命令，使云程序变得可执行，然后通过输入 cloud 运行它。清单 13 展示了这些步骤。

清单 **13.** 在 **Android** 手机上测试云

```
$ su
# cd data
# mkdir cloud
# cd cloud
# cp /sdcard/cloud .
# chmod 777 cloud
# ./cloud
# ps
```

```
--- Listing of resident daemons, and "./cloud" should be one of them. ---
```

按下 **Home** 键，启动浏览器，然后转到 URL http://localhost。您应当会在 Android 浏览器中看到微型云 的输出。您可以通过单击显示的链接来访问手机的文件系统。

接下来，如果您的 Android 手机 Wi-Fi 正在网络中运行，那么可以使用您的计算机来呼叫微型云。要实现此目的，您需要知道手机的 Wi-Fi IP 地址。可以使用许多方法得到此地址，包括检查您的 Wi-Fi 路由器的日志。另一种方法是返回 Terminal 程序并输入 `netstat -r`；您应当会看到类似清单 14 所示的条目。

清单 **14.** 使用 **Netstat** 获得手机的 **IP** 地址

```
# netstat -r
Proto Recv-Q Send-Q  Local Address    Foreign Address    ...
.
.
.
tcp        0      0  192.168.0.3:80   192.168.0.5:58744  ...
.
.
.
```

在浏览器的地址栏中输入 `http://192.168.0.3/`（手机的 IP 地址）。您应当会很快看到 Android 手机的文件系统清单，这是通过微型云显示的。

同时，要在您的手机中使用微型云浏览您创建的 Web 页面，只需要按住 **Home** 键并单击浏览器或编辑器来切换应用程序。如果单击浏览器的话，那么刷新菜单来查看已编辑的修改。通过反复地 “编辑/测试/编辑/测试” 并直到找到所需的内容，当在牙医办公室排队等候的时候，您就可以上网冲浪。

回页首

增强项目

可以使用许多方法对项目进行增强，但是您可以考虑以下这些建议：

- 添加一个名为 Java 的特定菜单项，该菜单项将自动导航到 Java 类文件所在的目录，允许您单击其中的任何类文件来进行浏览（类文件实际上是经过压缩的目录，因此可以使用超链接）。您可以对包含感兴趣的代码的任何其他特定目录实现这一功能。
- 添加一个 top-like 页面，它大约每分钟执行一次刷新并显示从 /proc 目录中读取的选项信息。
- 编写一些代码，使您能够从 Web 页面查询 SQLite 数据库。
- 所有基于 HTML 的演示可以放到 SD 卡中，并且手机的可移植性非常出众，因此通过使用大多数现代呈现平台（stage）中内置的 Web 浏览器，您可以在您的手机中显示自己的幻灯片演示。

当您决定对微型云进行一项改进时，执行下面三个一般步骤来添加功能：

1. 对 清单 6 中的 `buildbuf` 函数进行修改，输入一个表示您希望添加的功能的新菜单选择。
2. 修改 清单 9 中的 `child` 函数，从而可以为新功能提供服务。例如，了解 `About_` 菜单项如何在这些功能（`buildbuf` 和 `child`）中工作，您将看到向微型云添加新功能是多么简单。
3. 编写为菜单项提供服务的函数，并确保在 child 中插入对新函数的调用。

回页首

结束语

尽可能多地了解您所使用的设备始终是一件好事。此外，编写能够帮助您了解设备的工具也非常有趣。将一个微型云计算应用程序放到 Android 手机的内核中是一个有趣的、可以学到许多知识的过程，它将使您能够了解到许多隐藏在这些神奇设备内部的知识。