第 5 章　应用软件开发平台

有了操作系统这个软件运行协同用户操作平台，人们就可以使用计算机了。终端用户可以在上面运行应用程序，例如用文字处理软件编写文档、用播放器软件看电影、用电子表格软件管理自己的信息等。程序员可以利用其提供的系统调用接口编写应用程序，获得操作系统平台提供的服务。但是，对于程序员来说，仅有操作系统提供的系统调用接口是不行的，他们需要有一个能够编辑程序代码的软件组织源程序，需要有编译软件将用高级语言编写的程序转换成机器指令，以便其能被计算机执行，还需要有帮助他们发现和纠正程序中的逻辑错误的调试工具。除了编辑器、编译器、调试器这些基本工具之外，现代应用程序开发常常是基于某种共性软件设施进行的。那些软件设施为复杂应用系统的开发和运行提供了极大的便利，它们的表现形式一般为某种 API(Application Program Interface，应用程序接口)，即从应用程序中可以直接调用的函数库，Java 中的各种框架就可以看成这样的设施。这些实际上是支持代码重用的软件设施，加上各种工具就形成了应用软件开发平台。这个平台就好像是布满各种原材料和零配件的生产车间或装配场地，加上钳子、锤子等各种必需的工具就能形成一套完备的系统。

应用软件开发平台本身也是软件，其作用就是帮助应用软件设计人员更有效地开发出应用软件，它本身并没有直接的最终应用目的。有些开发平台是通用的，例如 CORBA；而有些是与特定领域相关的，例如 SAP，它能够有效地支持企业管理类软件的开发。本章介绍程序员开发应用软件时所用的应用软件开发平台。特别值得指出的是，随着开发平台的发展，人们将相关工具和可重用的代码统一组织起来，形成了集成开发环境，程序员在其中工作能够更自如地进行软件开发，大大提高了工作效率。

本章首先从程序设计语言说起，因为这是程序开发的基础，也是应用软件开发平台的功能得以体现的对象；然后简要介绍开发工具和集成开发环境中的一些基本概念，旨在让读者对应用软件开发平台形成一个总体轮廓。至于各种开发工具和集成开发环境的具体使用，则不属于本书的内容。

5.1　高级程序设计语言

现在人们多采用高级程序设计语言编写应用软件。不过，计算机世界中有几千种不同的高级语言。这些高级语言在语义、语法以及功能上都有着很大不同，因而它们也都有着各自不同的编译器。与之对应的就是程序员看到的不同的应用软件开发平台。因此，本章对应用软件开发平台的介绍首先从高级语言入手。

计算机世界中的高级语言非常多，这些语言产生的时间、背景不同，设计的目标也不同。程序员会根据所做项目的情况，有时还会根据自己的开发背景选择程序设计语言。经典的程序设计语言有 FORTRAN、Pascal、BASIC、C、COBOL、Ada、C ++ 、Java、Delphi、Python 等。在这些语言中，早期较重要的有 FORTRAN、Pascal、BASIC 以及 C 语言。而现在最常

用的是 Python、Java、C++ 以及 C。FORTRAN 是最早产生的高级程序设计语言，它是为工程计算而设计的，因而它非常适合处理公式以及进行各种数值计算。BASIC 是一种非常容易学的语言，它现在仍然在被使用，只不过用的是它的高级后代 Visual Basic。而 Pascal 是一门系统地体现结构化程序设计思想的高级语言。因此，它曾经作为各个大学计算机系必修课程的教学语言。后来的 Delphi 就是由 Pascal 语言发展而来的。C 语言是为了编写 UNIX 而设计的一种语言。因为是用来写操作系统的语言，所以它具有低级语言的一些特征，能够像汇编语言一样对硬件进行操作。但同时它也具有高级语言的特性，具有结构语句，也比较接近人的语言。因为这些特点，C 语言现在仍然是比较常用的一种程序设计语言。现在的操作系统（如 UNIX、Linux）就是用 C 语言编写的。现在 C 语言更多地被用在嵌入式程序设计等与底层硬件比较接近的应用软件的开发中。

C 语言属于结构化程序设计语言。人们把程序的任务分解为一个一个功能，分别加以实现，称之为函数，程序的主流程就是调用这些功能函数完成任务。这种程序设计模式也被称为面向过程的程序设计，因为程序的主体是一个求解任务的过程。但是，随着程序设计规模的扩大以及软件业的发展，人们的思想发生了变化。人们发现，如果把设计应用软件的事情看作在创造一个个对象或者说物体，那么就可以更好地模拟现实世界。程序设计就是设计一个个具有属性和行为的对象。应用软件因此不再是函数的集合体或者解题过程的描述，而是变成了对象的集合体。程序员在程序中定义对象以及对象的属性和行为，然后通过对象之间相互作用产生需要的结果。在这种思想指导下，产生了两个非常流行的程序设计语言——C++ 和 Java。

从名字上看就知道 C++ 和 C 是有关系的，而且好像应该是 C 语言的超级加强版，因为它有两个加号。事实也是这样的。C++ 保留了 C 语言的语法和语句。C 语言的程序在 C++ 环境中可以继续运行，这是因为在设计 C++ 时 C 语言已经非常流行，有不计其数的程序是用 C 语言开发的。C++ 比 C 语言要丰富很多，它增加了面向对象的一系列支持，如类、模板、名字空间、异常等概念，以及运算符重载、引用，还有在任何地方声明变量的能力等。因此，C++ 的功能更强大。

Java 是目前主流的开发语言之一，它也是一个面向对象的程序设计语言。不过，另一个追求目标使它与 C++ 有很大的不同。这个目标就是跨平台。我们知道，有很多种不同的操作系统平台，不同的操作系统提供的终端用户接口、程序员级接口以及底层的运行机制都是不一样的。因而在不同操作系统上运行的应用软件也是不同的，它们必须适应自己底层的平台环境。这些编译后的应用软件，甚至在源程序中，都带有与操作系统相关的元素，如对特定系统接口的调用。所以，通常在一个操作系统上设计开发的应用软件是不能在另一个操作系统上运行的。例如，Linux 操作系统的应用软件就不能在 Windows 操作系统平台中运行，反之亦然。而 Java 的跨平台目标就是要让应用软件可以在任意操作平台上运行。

Java 并没有采用像 C++ 那样的传统程序设计思路，即把高级语言源程序编译成可执行的机器指令代码文件，然后再执行。因为这样编译的结果是针对编译器所在的具体操作系统平台的，可执行文件也就被限制在特定的平台上了。为了实现跨平台的目标，Java 语言的源程序写完之后，会被转换成一种 Java 定义的中间状态的字节码形式。同时，Java 语言的设计人员还设计了一个在操作系统之上的称为 Java 虚拟机（Java Virtual Machine，JVM）的平台层。Java 虚拟机实际上也是一个程序，它运行在操作系统平台之上，其作用就是把

Java 字节码翻译成程序当时所在运行平台的机器指令形式并执行。因此，Java 程序必须在 Java 虚拟机上才能运行。也就是说，Java 字节码到不同运行平台的映射是由 Java 虚拟机完成的。Java 虚拟机屏蔽了不同运行平台的差异，让 Java 程序有一个统一的运行平台。可以看出，运行 Java 程序时，每条指令都是经过 Java 虚拟机的转换之后再执行的，这被称为解释执行。而 C++ 等语言编写的应用程序被编译后生成的可执行文件的代码都是机器指令，所以执行程序时，CPU 可以直接逐条完成指令。因此，Java 程序的执行要比其他编译后执行的程序慢，这是 Java 为跨平台付出的代价。

最后要特别说一下最近超级流行的一个“古老的”语言——Python。Python 是由 Guido Van Rossum 于 20 世纪年代末在荷兰国家数学和计算机科学研究所设计的。Python 是一种动态的、面向对象的脚本语言，是一门解释性高级编程语言。它最初被设计用于编写自动化脚本，就是 Shell 脚本。但随着版本的更新和语言新功能的添加，它被越来越多地用于独立项目和大型项目的开发。Python 源代码遵循 GPL 协议，由一个核心开发团队维护。Python 的优点是非常容易上手，关键字少，语法结构简单，但同时却有非常广泛的标准类库支持。尤其是随着人工智能研究的兴起，Python 对大多数模型都提供了非常好的支持。因此，最近几年 Python 已经上升为第一位的程序设计语言。

Python 也是跨平台的，这一点和 Java 一样。因此 Python 在 Linux、UNIX、Windows、嵌入式系统中都有很好的应用。Python 也可以嵌入 C\C++ 程序中作为脚本使用。

重点提示

目前常用的高级程序设计语言有 Python、Java、C++ 、C 等。
C 语言是面向结构的程序设计语言。
C++ 和 Java 都是面向对象的程序设计语言。
Java 有跨平台的特性。
Python 对大多数人工智能模型都提供了非常好的支持。

5.2 开发工具和集成开发环境

5.2.1 单独的开发工具

由于编写程序的需要，在程序设计的历史进程中，逐渐形成了各种供程序员使用的工具。这些工具中最主要的有 3 类，即用来编辑源程序的编辑软件、把高级语言编译成机器语言的编译软件以及调试程序的调试软件。这些工具构成了应用软件开发平台的基本元素。

不过，刚开始时应用软件开发平台中的这些软件工具都是独立的，它们之间并没有必然的联系。程序员需要自己决定使用哪个工具完成某个阶段的工作。某个阶段候选的工具也不只有一个。例如，编辑程序时，程序员可以选择 emacs，也可以选择 vi。实际上，有很多编辑软件都可以用来编辑程序。在这些编辑器中，有些是专门为编写程序而设计的。它们会对程序设计有特殊的支持，如用特殊的颜色显示程序设计语言中的特殊保留字。但有些编辑软件就是普通的编辑器，它们的目标就是帮助用户写一般的文档，就像 Windows 中的记事本软件一样。但是，只要程序员愿意，他们同样可以用这些编辑器输入程序。采用单独的开发工具，在程序员编辑完源程序后，他们需要把代码以某种扩展名的文件形式保存起来。

特殊的扩展名用来表明该源程序是用什么语言写的，例如.c 表示是 C 语言的源程序，这样做通常是后面要用到的编译程序的要求。接下来，程序员需要用其所在平台上的编译软件编译源程序。每种语言在支持的平台上都会有对应的编译器。例如，在 Linux 平台上，C 语言的编译器称为 gcc。程序编译成功后，并不意味着程序的运行结果是正确的。因为程序中可能存在逻辑性错误。为此，当程序并没有产生预期的结果时，程序员需要调试程序以确定到底哪里有问题。这时程序员会选择调试工具帮助他们解决问题。例如，如果在 Linux 系统上，程序员通常会选择一个称为 gdb 的调试器调试程序。尽管程序员在开发应用软件的过程中，每个阶段都有相应的工具可以使用，但是这些工具并没有被关联在一起。程序员必须知道他要用什么工具做下一步的工作。这就好像生产车间里放着很多可用的工具，但是，操作员必须自己决定接下来用什么来做事。使用单独的开发工具，程序员有很大的自由度，如果他们不喜欢某个调试器，那么完全可以换一个更好的。

因为单独的开发工具基本上都是早期出现的，所以多是基于文本命令界面的。这里要说的是，尽管这些工具出现得很早，但是，它们的功能一点儿也不差，而且这些工具仍处于不断发展进步之中，所以现在仍然有很多程序员喜欢使用这些工具。下面通过几个典型的工具体验一下这些单独的开发工具的特点以及在文本界面下开发应用软件的过程。

首先是编辑器，这里以 vi 为例。在 Linux 的 Shell 提示符后输入 vi，就进入了 vi 编辑器，如图 5-1 所示。vi 就是 visual，意思是马上就可以看到编辑的结果，这是相对于以前一些不能够马上见到编辑结果的编辑器来说的，当然现在绝大多数编辑器都支持所见即所得了。不过，在 vi 出现时这个特性还是比较先进的。

```
#include <stdio.h>

greeting()
{
        printf("Hello, world");
}

main()
{
        greeting();
}
```

图 5-1　vi 编辑器

vi 支持用不同的颜色表示程序中不同的内容，例如关键字 include 和其他语句的颜色就不同。函数 printf()的参数是一个用双引号括起来的字符串"hello, world"，这个字符串也用另一个颜色显示。这样程序员就可以看出双引号中间的字符串是什么，可以防止程序员忘记了写后面的双引号，造成程序的编译错误或者运行中的逻辑错误。vi 不支持鼠标，也没有命令菜单。它的命令和需要编辑的内容全是通过键盘输入的。为了区分键盘输入的内容是编辑命令还是被编辑的内容，vi 设置了命令和编辑两个模式。按 Esc 键即进入命令模式。在命令模式下，键盘的输入被解释成命令。例如，输入

```
:q
```

表示退出 vi，输入

```
:w
```

表示存盘。而按字母键 a 表示在当前光标后输入内容，输入这个命令后，vi 即进入编辑模式，如图 5-2 所示。这时屏幕下方的“——插入——”也提示了这时可以往文本中添加内容。例如，从键盘输入“：q”，这两个字符就会出现在编辑内容中，如图 5-2 所示。这时按 Enter 键，光标会移到下一行。接下来如果按 Esc 键再输入“：q”，就会看到这两个字符出现在屏

幕的下方，如图 5-3 所示，表明现在处于命令模式，输入了退出命令。如果接下来按 Enter 键，那么 vi 就会结束并退出。

图 5-2　vi 的编辑模式

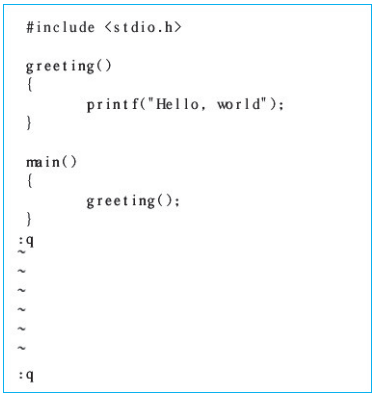

图 5-3　vi 的命令模式

vi 设置了很多命令。例如，在命令模式下，按 x 键会删除光标处的字符，输入 dd 会删除光标所在的行，输入 s 会删除光标处的字符并进入输入模式，输入 S 会删除光标所在的行并进入输入模式。在命令模式下，按 y 键(yank，复制)就可以复制光标所在行；移动光标，然后再按 p 键(put，放置)，就可以把这一行粘贴到光标所在位置。按斜杠(/)键，就可以在后面输入要查找的内容。例如，在图 5-4 中，在/后输入 printf 然后按 Enter 键，就可以看到如图 5-5 所示的查找结果。vi 还有很多命令，这里不一一介绍，可以在网上找到很多它的详细用法介绍，使用时可以查看。

图 5-4　vi 的查找命令

图 5-5　vi 的查找结果

vi 的不支持鼠标、通过键盘输入命令这两个特性让初次接触它的用户感觉很不方便。在某种程度上可以说 vi 的界面并不友好。这是因为它是特定条件下的产物，受制于当时的条件。不过，时至今日，vi 仍然没有退出舞台，那是因为键盘输入命令实际上也有它的优势，那就是程序员的手不用在鼠标和键盘之间切换。对于熟悉命令的人来说，这样确实很省力、方便，速度非常快。因而，现在仍然有很多程序员喜欢 vi，仍在使用它。

源程序编辑完毕之后，要想让它能在计算机上运行，必须转换成机器指令。在 Linux 的命令行界面上，最常用的编译器是 gcc。gcc 是 GNU Compiler Collection 的简称。假定前

面的文件被保存为 hello.c，如果要编译前面编辑的文件，那么可以输入以下命令：

```
gcc hello.c -o hello
```

命令中的-o 是 gcc 的选项，选项后的参数 hello 表示生成的可执行文件的名字为 hello，否则 gcc 会产生默认的文件 a.out。图 5-6 显示出一个错误提示，指出文件 hello.c 的 12 行有错误。这是因为前面在文件的末尾输入了“:q”。去掉这一行，重新编译，即成功地生成可执行文件 hello。运行 hello，屏幕上显示出“Hello, world”，如图 5-7 所示。这正是该程序的正确输出。

```
[root@zhanglivmware platform]# gcc hello.c -o hello
hello.c:12: parse error before ':' token
[root@zhanglivmware platform]#
```

图 5-6　gcc 编译文件 hello.c

```
[root@zhanglivmware platform]# gcc hello.c -o hello
hello.c:12: parse error before ':' token
[root@zhanglivmware platform]# vi hello.c
[root@zhanglivmware platform]# gcc hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
[root@zhanglivmware platform]#
```

图 5-7　成功编译文件 hello.c 并运行

除了-o 之外，gcc 还有很多选项供程序员使用。在 Shell 的命令提示符后输入

```
gcc -help
```

可以看到这些选项，如图 5-8 所示。程序员可以按照自己的需要选择这些选项，让 gcc 完成不同的工作。例如，在前面的编译过程中，gcc 实际上按顺序完成了预处理、编译、汇编、连接等几个工作。可以用选项把前面 gcc 一次完成的 4 个步骤分开来做。首先进行预处理。预处理过程主要处理源文件中的＃ifdef、＃include 和＃define 等命令。预处理使用的选项

```
[root@zhanglivmware platform]# gcc --help
Usage: gcc [options] file...
Options:
  -pass-exit-codes         Exit with highest error code from a phase
  --help                   Display this information
  --target-help            Display target specific command line options
  (Use '-v --help' to display command line options of sub-processes)
  -dumpspecs               Display all of the built in spec strings
  -dumpversion             Display the version of the compiler
  -dumpmachine             Display the compiler's target processor
  -print-search-dirs       Display the directories in the compiler's search path
  -print-libgcc-file-name  Display the name of the compiler's companion library
  -print-file-name=<lib>   Display the full path to library <lib>
  -print-prog-name=<prog>  Display the full path to compiler component <prog>
  -print-multi-directory   Display the root directory for versions of libgcc
  -print-multi-lib         Display the mapping between command line options and
                           multiple library search directories
  -print-multi-os-directory Display the relative path to OS libraries
  -Wa,<options>            Pass comma-separated <options> on to the assembler
  -Wp,<options>            Pass comma-separated <options> on to the preprocessor
  -Wl,<options>            Pass comma-separated <options> on to the linker
  -Xlinker <arg>           Pass <arg> on to the linker
  -save-temps              Do not delete intermediate files
  -pipe                    Use pipes rather than intermediate files
  -time                    Time the execution of each subprocess
  -specs=<file>            Override built-in specs with the contents of <file>
  -std=<standard>          Assume that the input sources are for <standard>
  -B <directory>           Add <directory> to the compiler's search paths
  -b <machine>             Run gcc for target <machine>, if installed
  -V <version>             Run gcc version number <version>, if installed
  -v                       Display the programs invoked by the compiler
  -###                     Like -v but options quoted and commands not executed
  -E                       Preprocess only; do not compile, assemble or link
  -S                       Compile only; do not assemble or link
  -c                       Compile and assemble, but do not link
  -o <file>                Place the output into <file>
  -x <language>            Specify the language of the following input files
```

图 5-8　gcc 的选项

是-E,表示只进行预处理,不做编译、汇编和连接。预处理过程输入的文件是 C 语言源文件。预处理过程会产生一个中间结果,用-o 选项指定存放预处理结果的文件名为 hello.i。这样预处理 hello.c 的具体命令即为

```
gcc -E hello.c -o hello.i
```

接下来对预处理的结果进行编译。编译过程的输入就是预处理产生的中间文件 hello.i。编译后生成汇编语言文件,所以文件的扩展名应该是.s。这时用 gcc 的选项-S,意思是只做编译,不做汇编和连接,命令如下:

```
gcc -S hello.i -o hello.s
```

下面对编译后的文件进行汇编,将其转换成机器指令文件。在 Linux 下机器语言文件的扩展名为.o。使用如下命令:

```
gcc -c hello.s -o hello.o
```

这时生成的文件就是机器指令文件了,但是它仍然不能够执行,因为其中有一些对库函数的调用需要进行连接。最后,用下面的命令将机器指令文件 hello.o 与其他机器指令文件或库文件连接成一个可执行的二进制代码文件:

```
gcc hello.o -o hello
```

上述整个过程如图 5-9 所示。

如果程序中存在问题,那么程序执行的结果就可能不像程序员期望的那样。这时程序员需要把问题找出来,调试工具会帮助程序员完成这件事。例如,如果在前面的 hello.c 文件中增加一段代码,希望能够输出 1～5 这几个数字的和,如图 5-10 所示。但是编译后程序的执行并没有输出预想的结果,如图 5-11 所示。

```
[root@zhanglivmware platform]# gcc -E hello.c -o hello.i
[root@zhanglivmware platform]# gcc -S hello.i -o hello.s
[root@zhanglivmware platform]# gcc -c hello.s -o hello.o
[root@zhanglivmware platform]# ./hello.o
bash: ./hello.o: 权限不够
[root@zhanglivmware platform]# gcc hello.o -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
[root@zhanglivmware platform]#
```

图 5-9　用 gcc 进行预处理、编译、汇编和连接

```
#include <stdio.h>

greeting()
{
        printf("Hello, world\n");
}

main()
{
        int sum,i;
        greeting();
        for(i=1;i<=5;i++){
                sum = sum +i;
        }
        printf("The sum of 1 to 5 is %d.\n",sum);

}
```

图 5-10　更改后的 hello.c

```
[root@zhanglivmware platform]# gcc hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
The sum of 1 to 5 is 1073828719.
[root@zhanglivmware platform]#
```

图 5-11　hello.c 编译后运行的结果

检查程序好像也没有问题。这时，程序员就可以让调试器帮忙了。在 Linux 命令行界面中常用的调试器是 gdb。要想使用 gdb，必须在编译源程序时使用-g 选项，这样编译过程中就会为调试产生必需的辅助信息。用-g 选项重新编译程序之后，就可以运行 gdb 对生成的可执行文件进行调试了，如图 5-12 所示。在图 5-12 中，运行 gdb 之后，在 gdb 的提示符(gdb)后输入 gdb 的调试命令 break，即

```
break 12
```

break 命令的作用是在程序中设置断点。上面这条命令的意思就是在 hello 程序的第 12 行设置断点。第 12 行就是 for 循环语句的那行。

接下来，用 gdb 命令 run 执行 hello 程序，如图 5-13 所示。程序将执行到第 12 行的断点处。这时输入 gdb 命令 next，让程序向下执行这条语句，即 for 语句，如图 5-14 所示。再用 gdb 命令 print 查看变量 i 的值，从图 5-14 中可以看到变量 i 的值是 1，这是一个正确的值。

```
[root@zhanglivmware platform]# gcc hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
The sum of 1 to 5 is 1073828719.
[root@zhanglivmware platform]# gcc -g hello.c -o hello
[root@zhanglivmware platform]# gdb hello
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(gdb) break 12
Breakpoint 1 at 0x8048355: file hello.c, line 12.
(gdb)
```

图 5-12　运行 gdb 调试 hello

```
[root@zhanglivmware platform]# gcc hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
The sum of 1 to 5 is 1073828719.
[root@zhanglivmware platform]# gcc -g hello.c -o hello
[root@zhanglivmware platform]# gdb hello
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(gdb) break 12
Breakpoint 1 at 0x8048355: file hello.c, line 12.
(gdb) run
Starting program: /root/platform/hello
Hello, world

Breakpoint 1, main () at hello.c:12
12              for(i=1;i<=5;i++){
(gdb)
```

图 5-13　执行 gdb 命令 run

接着再输出变量 sum 的值，如图 5-15 所示。可以看到，变量 sum 的值是一个很大的数，即 1 073 828 704。现在可以初步判断是变量 sum 的初值出了问题。

输入调试命令 next，让程序继续执行下一条指令，即 sum＝sum ＋ i。然后再输出变量 i 和 sum 的值，发现 i 变为 2，sum 的值被加了 1，如图 5-16 所示。这说明程序到这里的逻辑

```
(gdb) break 12
Breakpoint 1 at 0x8048355: file hello.c, line 12.
(gdb) run
Starting program: /root/platform/hello
Hello, world

Breakpoint 1, main () at hello.c:12
12              for(i=1;i<=5;i++){
(gdb) next
13                      sum = sum +i;
(gdb) print i
$1 = 1
(gdb)
```

图 5-14　执行 gdb 命令 next 和 print

```
(gdb) break 12
Breakpoint 1 at 0x8048355: file hello.c, line 12.
(gdb) run
Starting program: /root/platform/hello
Hello, world

Breakpoint 1, main () at hello.c:12
12              for(i=1;i<=5;i++){
(gdb) next
13                      sum = sum +i;
(gdb) print i
$1 = 1
(gdb) print sum
$2 = 1073828704
(gdb)
```

图 5-15　查看变量 sum 的值

是正确的。用 gdb 命令 quit 结束调试过程。重新用 vi 打开 hello.c 源程序，发现变量 sum 没有赋初值就使用了。因此，在 for 循环之前增加给 sum 赋初值 0 的语句，如图 5-17 所示。然后再编译、执行，发现程序的执行结果就是正确的了，如图 5-18 所示。其实，这个例子如果不用 gdb 帮忙，程序员也可能会找出问题，这里只是为了说明 gdb 的功能。对于复杂的程序，程序员没有 gdb 的帮助就会很难或者花费很多时间才能找到问题的所在。

```
12              for(i=1;i<=5;i++){
(gdb) next
13                      sum = sum +i;
(gdb) print i
$1 = 1
(gdb) print sum
$2 = 1073828704
(gdb) next
12              for(i=1;i<=5;i++){
(gdb) print i
$3 = 1
(gdb) next
13                      sum = sum +i;
(gdb) print i
$4 = 2
(gdb) print sum
$5 = 1073828705
(gdb)
```

图 5-16　执行下一条指令

```
int sum,i;
greeting();

sum=0;
for(i=1;i<=5;i++){
        sum = sum +i;
}
printf("The sum of 1 to 5 is %d.\n",sum);
```

图 5-17　修改 hello.c 源程序

```
(gdb) next
13                      sum = sum +i;
(gdb) print i
$4 = 2
(gdb) print sum
$5 = 1073828705
(gdb) q
The program is running.  Exit anyway? (y or n) y
[root@zhanglivmware platform]# vi hello.c
[root@zhanglivmware platform]# gcc -g hello.c -o hello
[root@zhanglivmware platform]# ./hello
Hello, world
The sum of 1 to 5 is 15.
[root@zhanglivmware platform]#
```

图 5-18　重新编译、执行 hello 程序

单独的开发工具赋予了程序员很大的灵活性，他们可以按照自己的喜好选择工具完成应用软件的开发。不过，这些单独的开发工具一般都是在文本命令行界面下的。因此，一般经验比较丰富、接触计算机较早的程序员采用这些工具的比较多，因为他们开始接触计算机时，计算机以文本命令行界面为主，因而他们比较习惯这种界面。而对于新生代的程序员来说，他们更喜欢图形界面的操作平台。让他们自己选择工具，一些人会不知所措。

重点提示

编辑、编译、调试工具是应用软件开发平台的基本元素。
单独的开发工具可以让程序员更自由地选择自己喜欢的工具。
单独的开发工具多是基于操作系统的文本命令界面的。
Linux 上最常见的编辑、编译、调试工具分别是 vi、gcc 和 gdb。

5.2.2　集成开发环境

为了克服单独的开发工具的缺点，不再让程序员自己寻找和组合开发工具，人们把软件开发过程中必需的工具整合在一起，形成了集成开发环境。这样，程序员只要启动一个集成环境软件就可以在其中完成全部的程序开发步骤了。这就好像把生产车间里面零乱的工具组装成一个机床。在程序设计的历史中出现了很多集成开发环境。这是因为有很多种不同的程序设计语言，而一般针对某一种语言，在每个操作系统平台上都会有一个或者几个集成开发环境。例如，在 Windows 操作系统上，C++ 的集成开发环境有 Visual C++、Turbo C/C++、C++ Builder、Visual Studio Code、Visual Studio、Qt Creator、CodeBlocks、Dev C++ 等，Java 的集成开发环境有 JDK、Eclipse、JBuilder、WebSphere Studio、NetBeans、JDeveloper、IntelliJ IDEA、Java Workshop、Sun Java Studio 等。其中一些开发平台可以支持多种语言，如 Eclipse、Qt Creator 等。

这些集成开发环境的一些早期版本也是在文本命令行界面的操作平台上出现的，如 Turbo C。后来随着图形界面的普及，这些集成开发环境就多是图形界面的了。这些集成开发环境在界面和用法上有一些不同，但是主要界面框架和基本功能是一致的。它们都支持编辑、编译、调试、运行等基本功能，这些功能的用法也基本类似。这就好像各种汽车的样子和功能会有一些不同，但是，基本的组成都是车体和车轮，主要功能都是能在路上行驶。车的驾驶方式也不相同，不过，总的来说，都是通过控制方向盘、离合、刹车等装置实现驾驶。集成开发环境与此非常类似。因此，只要熟悉了一种集成开发环境，就很容易快速掌握另一种。

图 5-19 是集成开发环境 Qt Creator 的界面。Qt Creator 是一个跨平台的集成开发环境，可在 Windows、Linux 和 macOS 桌面操作系统上运行。在 Qt Creator 高级代码编辑器上可以用 C++、QML、JavaScript、Python 和其他语言编写代码。

图 5-19 反映出目前多数集成开发环境的面貌。界面中间的窗口是程序的编辑窗口，程序员可以在其中输入代码。左侧的窗口会显示出当前项目（或解决方案）涉及的各种文件。除此之外，左侧窗口中通常还有一些标签窗口，用于显示项目中包含的类以及类的属性等内容。界面下方是输出窗口，这里可以输出程序编译和连接的结果、程序调试的中间结果等。界面上方是菜单栏和工具栏。菜单栏和工具栏反映了开发环境的主要功能，例如从菜单栏