```
4   {
5       int fd1, fd2;
6       char c;
7
8       fd1 = Open("foobar.txt", O_RDONLY, 0);
9       fd2 = Open("foobar.txt", O_RDONLY, 0);
10      Read(fd2, &c, 1);
11      Dup2(fd2, fd1);
12      Read(fd1, &c, 1);
13      printf("c = %c\n", c);
14      exit(0);
15  }
```

10.10　标准 I/O

C 语言定义了一组高级输入输出函数，称为标准 I/O 库，为程序员提供了 Unix I/O 的较高级别的替代。这个库(libc)提供了打开和关闭文件的函数(fopen 和 fclose)、读和写字节的函数(fread 和 fwrite)、读和写字符串的函数(fgets 和 fputs)，以及复杂的格式化的 I/O 函数(scanf 和 printf)。

标准 I/O 库将一个打开的文件模型化为一个*流*。对于程序员而言，一个流就是一个指向 FILE 类型的结构的指针。每个 ANSI C 程序开始时都有三个打开的流 stdin、stdout 和 stderr，分别对应于标准输入、标准输出和标准错误：

```
#include <stdio.h>
extern FILE *stdin;    /* Standard input (descriptor 0) */
extern FILE *stdout;   /* Standard output (descriptor 1) */
extern FILE *stderr;   /* Standard error (descriptor 2) */
```

类型为 FILE 的流是对文件描述符和*流缓冲区*的抽象。流缓冲区的目的和 RIO 读缓冲区的一样：就是使开销较高的 Linux I/O 系统调用的数量尽可能得小。例如，假设我们有一个程序，它反复调用标准 I/O 的 getc 函数，每次调用返回文件的下一个字符。当第一次调用 getc 时，库通过调用一次 read 函数来填充流缓冲区，然后将缓冲区中的第一个字节返回给应用程序。只要缓冲区中还有未读的字节，接下来对 getc 的调用就能直接从流缓冲区得到服务。

10.11　综合：我该使用哪些 I/O 函数？

图 10-16 总结了我们在这一章里讨论过的各种 I/O 包。

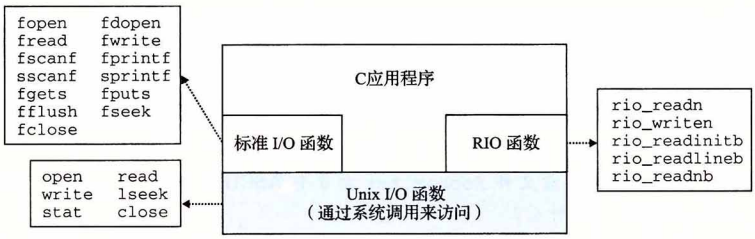

图 10-16　Unix I/O、标准 I/O 和 RIO 之间的关系