旁注　关于确定大小的整数类型的更多内容

对于某些程序来说，用某个确定大小的表示来编码数据类型非常重要。例如，当编写程序，使得机器能够按照一个标准协议在因特网上通信时，让数据类型与协议指定的数据类型兼容是非常重要的。我们前面看到了，某些 C 数据类型，特别是 long 型，在不同的机器上有不同的取值范围，而实际上 C 语言标准只指定了每种数据类型的最小范围，而不是确定的范围。虽然我们可以选择与大多数机器上的标准表示兼容的数据类型，但是这也不能保证可移植性。

我们已经见过了 32 位和 64 位版本的确定大小的整数类型(图 2-3)，它们是一个更大数据类型类的一部分。ISO C99 标准在文件 stdint.h 中引入了这个整数类型类。这个文件定义了一组数据类型，它们的声明形如 int*N*_t 和 uint*N*_t，对不同的 N 值指定 N 位有符号和无符号整数。N 的具体值与实现相关，但是大多数编译器允许的值为 8、16、32 和 64。因此，通过将它的类型声明为 uint16_t，我们可以无歧义地声明一个 16 位无符号变量，而如果声明为 int32_t，就是一个 32 位有符号变量。

这些数据类型对应着一组宏，定义了每个 N 的值对应的最小和最大值。这些宏名字形如 INT*N*_MIN、INT*N*_MAX 和 UINT*N*_MAX。

确定宽度类型的带格式打印需要使用宏，以与系统相关的方式扩展为格式串。因此，举个例子来说，变量 x 和 y 的类型是 int32_t 和 uint64_t，可以通过调用 printf 来打印它们的值，如下所示：

```
printf("x = %" PRId32 ", y = %" PRIu64 "\n", x, y);
```

编译为 64 位程序时，宏 PRId32 展开成字符串“d”，宏 PRIu64 则展开成两个字符串“l”“u”。当 C 预处理器遇到仅用空格(或其他空白字符)分隔的一个字符串常量序列时，就把它们串联起来。因此，上面的 printf 调用就变成了：

```
printf("x = %d, y = %lu\n", x, y);
```

使用宏能保证：不论代码是如何被编译的，都能生成正确的格式字符串。

关于整数数据类型的取值范围和表示，Java 标准是非常明确的。它要求采用补码表示，取值范围与图 2-10 中 64 位的情况一样。在 Java 中，单字节数据类型称为 byte，而不是 char。这些非常具体的要求都是为了保证无论在什么机器上运行，Java 程序都能表现地完全一样。

旁注　有符号数的其他表示方法

有符号数还有两种标准的表示方法：

反码(Ones' Complement)：除了最高有效位的权是 $-(2^{w-1}-1)$ 而不是 -2^{w-1}，它和补码是一样的：

$$B2O_w(\vec{x}) \doteq -x_{w-1}(2^{w-1}-1)+\sum_{i=0}^{w-2}x_i2^i$$

原码(Sign-Magnitude)：最高有效位是符号位，用来确定剩下的位应该取负权还是正权：

$$B2S_w(\vec{x}) \doteq (-1)^{x_{w-1}} \cdot \left(\sum_{i=0}^{w-2}x_i2^i\right)$$

这两种表示方法都有一个奇怪的属性，那就是对于数字 0 有两种不同的编码方式。这两种表示方法，把[00…0]都解释为 +0。而值 −0 在原码中表示为[10…0]，在反码中表示为[11…1]。虽然过去生产过基于反码表示的机器，但是几乎所有的现代机器都使用补码。我们将看到在浮点数中有使用原码编码。