层，所有这些工作称为**多路复用**（multiplexing）。值得注意的是，图 3-2 中的中间那台主机的运输层必须将从其下的网络层收到的报文段分解后交给其上的 P_1 或 P_2 进程；这一过程是通过将到达的报文段数据定向到对应进程的套接字来完成的。中间主机中的运输层也必须收集从这些套接字输出的数据，形成运输层报文段，然后将其向下传递给网络层。尽管我们在因特网运输层协议的环境下引入了多路复用和多路分解，认识到下列事实是重要的：它们与在某层（在运输层或别处）的单一协议何时被位于接下来的较高层的多个协议使用有关。

为了说明分解的工作过程，可以再以前面一节的家庭进行类比。每一个孩子通过他们的名字来标识。当 Bill 从邮递员处收到一批信件，并通过查看收信人名字而将信件亲手交付给他的兄弟姐妹们时，他执行的就是一个分解操作。当 Ann 从兄弟姐妹们那里收集信件并将它们交给邮递员时，她执行的就是一个多路复用操作。

既然我们理解了运输层多路复用与多路分解的作用，那就再来看看在主机中它们实际是怎样工作的。通过上述讨论，我们知道运输层多路复用要求：①套接字有唯一标识符；②每个报文段有特殊字段来指示该报文段所要交付到的套接字。如图 3-3 所示，这些特殊字段是**源端口号字段**（source port number field）和**目的端口号字段**（destination port number field）。（UDP 报文段和 TCP 报文段还有其他的一些字段，这些将在本章后继几节中进行讨论。）端口号是一个 16 比特的数，其大小在 0 ~ 65535 之间。0 ~ 1023 范围的端口号称为**周知端口号**（well-known port number），是受限制的，这是指它们保留给诸如 HTTP（它使用端口号 80）和 FTP（它使用端口号 21）之类的周知应用层协议来使用。周知端口的列表在 RFC 1700 中给出，同时在 http://www.iana.org 上有更新文档［RFC 3232］。当我们开发一个新的应用程序时（如在 2.7 节中开发的一个应用程序），必须为其分配一个端口号。

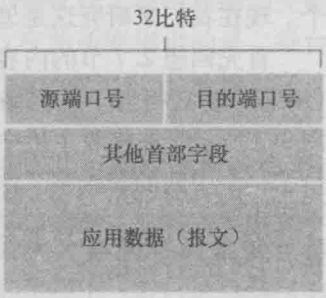

图 3-3 运输层报文段中的源与目的端口字段

现在应该清楚运输层是怎样能够实现分解服务的了：在主机上的每个套接字能够分配一个端口号，当报文段到达主机时，运输层检查报文段中的目的端口号，并将其定向到相应的套接字。然后报文段中的数据通过套接字进入其所连接的进程。如我们将看到的那样，UDP 基本上是这样做的。然而，也将如我们所见，TCP 中的多路复用与多路分解更为复杂。

1. 无连接的多路复用与多路分解

2.7.1 节讲过，在主机上运行的 Python 程序使用下面一行代码创建了一个 UDP 套接字：

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

当用这种方式创建一个 UDP 套接字时，运输层自动地为该套接字分配一个端口号。特别是，运输层从范围 1024 ~ 65535 内分配一个端口号，该端口号是当前未被该主机中任何其他 UDP 端口使用的号。另外一种方法是，在创建一个套接字后，我们能够在 Python 程序中增加一行代码，通过套接字 bind() 方法为这个 UDP 套接字关联一个特定的端口号（如 19157）：

```
clientSocket.bind(('', 19157))
```