R 数据导入和导出

版本：2.2.1（2005 年 12 月 20 日）
R 开发核心小组

目录
致谢

致谢

手册中关系数据库内容部分基于 Douglas Bates 和 Saikat DebRoy 的早期手册。本手册的主要作者是 Brian Ripley。

许多志愿者为手册中使用的软件包作出了贡献。这些涉及的软件包的主要作者是：

CORBA Duncan Temple Lang
foreign Thomas Lumley, Saikat DebRoy, Douglas Bates, Duncan，Murdoch and Roger Bivand
hdf5 Marcus Daniels
ncdf David Pierce
ncvar Juerg Schmidli
RMySQL David James and Saikat DebRoy
RNetCDF Pavel Michna
RODBC Michael Lapsley and Brian Ripley
RSPerl Duncan Temple Lang
RSPython Duncan Temple Lang
SJava John Chambers and Duncan Temple Lang
XML Duncan Temple Lang

Brian Ripley 是支持连接（connection）的作者。

第一章：概述

统计分析系统读入数据和输入结果报告到其他系统是让人沮丧的任务，会花费比统计分析本身更多的时间，虽然读者会发现统计分析要更加吸引人。

本手册描述了通过 R 本身或者通过可来自 CRAN 的软件包读写数据的机制。虽然涉及到的软件包有的还在开发中，但是他们提供了有用的功能。

除非特别说明，手册中描述的内容适用于任何平台中的 R。

一般来说，类似 R 这类统计分析系统对大批量的数据不是特别合适。这个方面其他系统比 R 更加合适，手册一些地方提到说与其增加 R 的功能，不如让其他系统进行这项工作。（比如，Therneau 和 Grambsch (2000)说他们习惯在 SAS 中操作数据，然后使用 S 的 survival 分析。）最近一个软件包允许以 Java, perl 和 python 等语言直接整合 R 代码来开发，利用这些语言提供的功能也许适合些。（从 Omegahat 工程 http://www.omegahat.org 上可以得到 See the SJava, RSPerl and RSPython 软件包）

R 和 S 类似有 Unix 可重用的小工具的传统，在导入数据前和输出结果后可以使用 awk 和 perl 来处理。在 Becker, Chambers 和 Wilks (1988, Chapter 9)的书中有一个例子就是通过 Unix 的工具来检验和操作输入到 S 的的数据。R 自身使用 perl 来操作起帮助文件，可以使用 read.fwf 函数来调用 perl 脚本，知道停止 perl 为止。Unix 传统的工具目前有很多，包括适用于 Windows 平台的。

1.1 导入

早期导入到 R 的数据是简单的文本格式的文件，中小规模的数据通常是可以接受的。导入文本文件的基本函数是 scan，随后会有很多更加好用的函数将在第二章中讨论（[Spreadsheet-like data], page 5）。

然而所有统计顾问都会遇到客户提交的包含权限设置的二进制数据的软盘或者 CD，比如说 Excel 电子表格或者 SPSS 数据文件。通常，可以导出为文本格式的文件（这样统计顾问就获得了他们电脑上最普通应用数据格式文件的备份。）但是，有时候这是不可能的，第三章（[Importing from other statistical systems], page 11）讨论了通过 R 直接读取这些文件的机制。读取 Excel 电子表格的方法在第八章（[Reading Excel spreadsheets], page 25）中做了总结。

有时候，数据压缩以后成为加速读取的二进制文件。我们有时候把图像数据通过这种方式来处理，在内存中以比特流的形式存在。这种形式的文件读取在第五章和第六章第五节（[Binary files], page 18 and Section 6.5 [Binary connections], page 21）中讨论。

1.2 导出到文本文件

从 R 中导出结果一般来说没什么争议，但是还是有些缺陷。通常，文本格式的文件是最方便的交换媒介。（如果需要二进制文件，参照第五章[Binary files], page 18.）

函数 cat 成为结果导出函数的基础。cat 函数通过一个函数作为参数，附带一些其他参数使得通过正确调用 cat 函数得到一个文本文件。更好的是，可以多次建立连接，打开一个文件用来写入或添加，然后关闭文件。

常见的工作是以矩阵的形式把一个矩阵或者数据框写入到文件中，有时候还带上行和列的名称。这可以通过 write.table 和 write 函数来完成。函数 write 仅写出一个制定列数的矩阵或者向量。write.table 函数更加方面，可以写出一个数据框（或者可以转化为数据库形式的对象），并带有行列标记。写出数据框到文本文件还有一些问题需要考虑。

1.精度问题

通过这些函数，大多数实数或者复数转换为完全的精度，write 函数得到的精度依赖于 option（digits）的当前设置。为了进一步细致的控制，可以使用 format 函数来一列一列的操作数据框。

2.首行问题

R 默认首行为变量名称行，于是文件内容常有如下格式：

```
          dist    climb    time
Greenmantle  2.5     650      16.083
……
```

如果 write.table 函数的 col.names 参数为 NA 的话，其他一些系统需要输入变量名称行。Excel 中就是这样的情况。

3.分隔符

在英语国家中当逗号不出现在任何字段中的时候，文件中常见分隔符是逗号。这些文件被称为 CSV(comma separated values)文件，封装好的函数 write.csv 提供了一些合适的默认选项来读取数据。一些情况中，逗号被用作进制符号(在 write.table 函数中设置参数 dec = ",")，这时候使用 write.csv2 默认参数来读取数据，使用分号作为分隔符。使用分号或者 tab（sep="\t"）设置是最安全的选择。

4.缺失值

默认情况下，缺失值的输出为 NA，但是可以通过参数 na 的设置来改变。请注意，wrtie.table 函数中把 NaNs 当作 NA 处理，但是 cat 和 write 函数中不是如此。

5.引号

默认情况下，字符串都有引号（包括行和列的名称）。参数 quote 决定了字符和因子变量的引号形式。字符串中含有引号的时候需要注意，三种有用的形式如下：

```
> df <-data.frame(a = I("a \" quote"))
> write.table(df)
"a"
"1" "a \" quote"
> write.table(df, qmethod = "double")
"a"
"1" "a "" quote"
> write.table(df, quote = FALSE, sep = ",")
a
1,a " quote
```

escape 的第二种形式在电子表格较常用。

MASS 包中的 write.matrix 函数提供了写出矩阵格式数据的特别接口，选择块的形式从而节省了内存消耗。

可以使用 sink 函数把标准的 R 输出写到一个文件，从而获得打印的说明。这不是通常最有效的方式，options(width)设置也许需要增加。

foreign 包中的 write.foreign 函数通过使用 write.table 函数产生一个文本文件，同时也可以给出读取这些文本文件到其他统计软件包需要的代码。现在可以支持数据输出到 SPSS 和 Stata 的情况。

1.3 XML

从文本文件中读取数据的时候，用户需要知道如何定制产生文件的一些转换设置，比如说评注字符，是否需要首行（名称），值的分隔符，缺失值的表示等等，这些内容在 1.2 节（[Export to text files], page 3）中做了说明。一种可以不仅用来保存内容，而且可以提供内容结构的标记性语言可以让文件自我说明，一次不需要提供细节就可以被软件读取数据了。

XML（可扩展性标记语言）可以提供这种结构，不仅可以提供技术的数据内容，而且可以提供负责的数据结构。XML 极其流行并且成为通行标记和交换语言的标准。可以在不同场合下描述地理学的数据，比如地图，图表展示和数学内容等等。

XML 包提供了读写 XML 文件通用的工具，在 R 和 S-PLUS 中都可以方便的使用这项技术。一些人展现了如何使用 XML 技术，同时也包括其他内容，去展示不同应用软件之间可以共同使用的数据集，存储 R 和 S-PLUS 的对象，使得在两个系统中都可用；通过 SVG（Scalable Vector Graphics, XML 的同义词）展示作图，表达函数文件；生成包括文本、数据和代码的动态分析报告。

XML 包的功能超出了本手册的范围，可以在 http://www.omegahat.org/RSXML 这个网页得到 XML 包的详细资料和例子。CRAN 中的 StatDataML 包就是基于 XML 的一个例子。

第二章 类电子表格数据

在 1.2 章节中（[Export to text files], page 3），我们看到电子表格形式的文本文件的几种情况，其中的数据是矩阵形式的，有的还有行和列的名称。本章节将要讨论读取这种格式的数据到 R 中。

2.1 read.table 函数的几种形式

read.table 函数是读取矩阵形式数据的最好方法。其他一些函数是调用了 read.table 函数加上了一些默认参数。需要注意的是，read.table 函数读取大数值矩阵是缺乏效率的，参见下面提到的 scan 函数。

下面是一些需要注意的方面：

1、编码:如果文件中含有非 ASCII 编码的字段，必须保证以正确的编码形式读入。这主要出现在 UTF-8 的情况下读取 Latin-1 文件格式，可以通过如下方式实现 read.table(file("file.dat", encoding="latin1"))。这种方法可以在 Latin-1 名字的任何情况下成功。

2、首行：我们推荐显式的指定首行参数，依照惯例，首行表示列名称而非行名称，于是出现了比其他行少一个字段的情况。（在 R 中，设置参数 header = TRUE）。如果文件中有

首个字段作为行名称，可以通过如下方式实现 read.table("file.dat", header = TRUE, row.names = 1)。列名称可以显式的通过 col.names 参数实现，显式的名称将不考虑首行。

3、分隔符：一般来说，可以看到文件内容就知道字段之间的分隔符，但是字段间有空格的时候，可能是缺失的参数 sep = ""，可以代表任意间隔（空格、制表符或者回车）作为分隔符。请注意，选择的分隔符会影响到带有引号的字符串。如果在制表符分隔符的文件中包含空字段，请使用 sep = "\t"作为参数。

4、引号：默认情况下，字符串使用单引号或者双引号，这时候所有字符中的引号都作为一部分来匹配。引号的有效设置通过 quote 参数来控制。 sep ="\n"默认的改变 quote = ""。如果没有指定分隔符，没有被引字符串前面使用 c 语言形式的‘\’。如果指定了分隔符，被引用字符串中的引号在电子表格文件中连续两次出现当作通常的一个字符。比如，read.table("testfile", sep = ",")方式可以读取’One string isn’’t two’,"one more" ，而不会对缺省的分隔符产生作用。

5、缺失值：默认情况下，包含 NA 的字符代表缺失值，不过可以通过 na.strings 参数来改变设置，na.strings 是包含一个和几个表示缺失值的字符。数值列中空值也被视作缺失值。在数值列中，NaN、Inf 和-Inf 是合法的。

6、尾行：通常从电子文档格式文件中导入的数据的时候，用参数 fill=TRUE 省略尾部空字段。

7、字符型字段中的空格:如果指定了分隔符，字符型字段中前后空格作为字段的部分存在，若想去掉这些空格，使用参数 strip.white=TRUE.

8、空白行：缺失情况下，read.table 函数略过空白行。可以通过参数 blank.lines.skip=FALSE 来改变设置，这是需要配套使用参数 fill=TRUE。

9、变量的类：除非你有特别的指定，read.table 函数为数据框中的每一个变量自动选择一个合适的类。遵照如下的顺序：logical, integer, numeric and complex，略过无法转化的部分。如果以上所有都失败了，变量转化为因子（factor）。参数 colClasses 和 as.is 提供了更强的控制，as.is 禁止字符型向量转化为因子。使用 colClasses 参数可以在输入数据的时候为各列指定类。需要注意的是，colClasses 和 as.is 参数用来指定各列，不是针对每个变量，于是包含了行名的那一列（如果这列存在的话）。

10、注解：缺省情况下，read.table 函数使用‘#’作为注释字符，如果被读入了，该行剩下的部分将被省略掉（除了使用引号的以保外）。空白行和注释行被视为空行。如果已知数据文件中没有注视，那么可以设置 comment.char=""，也许速度会更快。

11、Escapes：很多操作系统中，在文本文件中使用反斜杠作为 escape 字符，但是 Windows 操作系统中不是如此（而是使用反斜杠作为文件路径名的一部分）。在 R 中是否使用这样的惯例风格是可选的。在 read.table 和 scan 两个函数中，有一个 allowEscapes 的参数。从 R2.2.0 开始，缺省设置为 false，反斜杠被解释为 excape 符号（在上述情况下）。如果设置为 true，就解释为 C 语言风格的 escape 字符，称为控制字符，可以表示类似\a, \b, \f, \n, \r, \t, \v 以及八进制、十六进制\040 和\0x2A 的情况。

通用函数 read.csv 和 read.delim 给 read.table 函数提供了恰当的参数，用来读取英语情况下的 CSV 和制表符分割文件。而 read.csv2 和 read.delim2 函数提供了使用逗号作为小数点分割情况下读取类似文件的功能。

如果 read.table 函数中参数设置不正确的时候，出错提示信息通常有如下形式：
Error in scan(file = file, what = what, sep = sep, :

line 1 did not have 5 elements

或者

Error in read.table("files.dat", header = TRUE) :
more columns than column names

这提供了寻找错误而需要的足够信息，同时 count.fields 这个辅助函数可以用来做进一步的审核。

在读取大数据的时候，效率是重要的。通过设置 comment.char=""，把每一列通过 colClasses 为一个原子型向量（logical, integer, numeric, complex, character 或者 raw），设置 nrows——读取的行数——做一个恰当的估计要比不做任何设置好，这些都有利于提高数据读取效率。可以看下面的例子。

2.2 固定宽度格式的文件

有时候数据文件中的字段没有分隔符，但是字段实现制定了列的情况。这在打卡的时代是常见的事情，现在有时候也用来节省文件的空间。

通过指定一个包含字符宽度的向量，read.fwf 函数提供读取这种文件的简单途径。这个函数把整个文件作为整行读入内存，拆分字符，写出到一个临时的制表符分割的文件，然后调用 read.table 函数。对小文件而言这是尚可的办法，但是对任何复杂的情况，我们推荐使用 perl 等语言预处理一下文件转化格式。

read.fortran 函数是另外一个读取固定格式文件的函数，使用了 Fortran 风格的列确定方法。

2.3 直接使用 scan 函数

read.table 和 read.fwf 两个函数都通过 scan 函数来读取文件，然后处理 scan 函数得到的结果。这非常方便，不过有时候直接使用 scan 函数要好些。

函数 scan 有太多的参数，很多参数已经通过 read.table 函数介绍过了。最主要的参数是 what，它用来制定读取文件中变量的类型的列表。如果这个列表命名了，这些命名就会作用到返回列表的组成部分。类型可以是数值（numeric）,字符（character）或者复数（complex）等，通常可以通过例子来指定，比如 0，“”，0i 等。比如说：

cat("2 3 5 7", "11 13 17 19", file="ex.dat", sep="\n")
scan(file="ex.dat", what=list(x=0, y="", z=0), flush=TRUE)

会返回一个包含三个组建的列表，并且不读取文件中的第四列。

如果想要读取整行整行的数据便于进一步的处理，readLines 函数可能更方便。

scan 函数一个通常的用法是读取大的矩阵。假如文件 matrix.dat 恰好包含 200×2000 的矩阵。于是可以使用

A <-matrix(scan("matrix.dat", n = 200*2000), 200, 2000, byrow = TRUE)

一个测试表明该过程需要 1 秒钟（在 Linux 下面，相同硬件配置的 Windows 下需要 3 秒钟），而

A <-as.matrix(read.table("matrix.dat"))

需要 10 秒钟（而且需要更多的内存）。

```
A <-as.matrix(read.table("matrix.dat", header = FALSE, nrows = 200,
comment.char = "", colClasses = "numeric"))
```

则需要 7 秒钟。差异主要是因为读取 2000 分离短列的开销：当长度为 2000 的时候，scan 函数需要 9 秒钟，read.table 函数使用有效设置后（特别的，指定 colClasses）需要 18 秒钟，原始不设置情况下需要 125 秒钟。

请注意所需的时间和读取数据的类型相关，考虑读取百万个整数的情况：

```
writeLines(as.character((1+1e6):2e6), "ints.dat")
xi <-scan("ints.dat", what=integer(0), n=1e6) # 0.77s
xn <-scan("ints.dat", what=numeric(0), n=1e6) # 0.93s
xc <-scan("ints.dat", what=character(0), n=1e6) # 0.85s
xf <-as.factor(xc) # 2.2s
DF <-read.table("ints.dat") # 4.5s
```

在百万的小编码集的情况下：

```
code <-c("LMH", "SJC", "CHCH", "SPC", "SOM")
writeLines(sample(code, 1e6, replace=TRUE), "code.dat")
y <-scan("code.dat", what=character(0), n=1e6) # 0.44s
yf <-as.factor(y) # 0.21s
DF <-read.table("code.dat") # 4.9s
DF <-read.table("code.dat", nrows=1e6) # 3.6s
```

同时需要注意的是，花费的时间和操作系统密切相关（基础的读取任务在 Windows 下花费的时间至少是 Linux 下的两倍），以及和垃圾回收的情况有关。

2.4 改变数据形状

有时候，电子表格数据是紧凑型的。每个对象跟着全部的观测值。在 R 的建模函数中，需要观测值是独立的一列。考虑一个来自于 MRI 的脑测试样本数据：

Status	Age	V1	V2	V3	V4	P	23646
45190	50333	55166	56271	CC	26174	35535	38227
37911	41184	CC	27723	25691	25712	26144	26398
CC	27193	30949	29693	29754	30772	CC	24370
50542	51966	54341	54273	CC	28359	58591	58803
59435	61292	CC	25136	45801	45389	47197	47126

有两个 covariate，四次测量。数据从 Excel 文件‘mr.csv’中输出。

我们可以通过 stack 函数来操作数据给出单一相应。

```
zz <-read.csv("mr.csv", strip.white = TRUE)
zzz <-cbind(zz[gl(nrow(zz), 1, 4*nrow(zz)), 1:2], stack(zz[, 3:6]))
```

结果为:

	Status	Age	values	ind

X1	P	23646	45190	V1
X2	CC	26174	35535	V1
X3	CC	27723	25691	V1
X4	CC	27193	30949	V1
X5	CC	24370	50542	V1
X6	CC	28359	58591	V1
X7	CC	25136	45801	V1
X11	P	23646	50333	V2

……

而函数 unstack 是对应操作的函数，在导出数据时可能有用。另外一个完成此项任务的函数是 reshape 函数，通过

```
reshape(zz, idvar="id",timevar="var",
varying=list(c("V1","V2","V3","V4")),direction="long")
```

得到

	Status	Age	var	V1	id
1.1	P	23646	1	45190	1
2.1	CC	26174	1	35535	2
3.1	CC	27723	1	25691	3
4.1	CC	27193	1	30949	4
5.1	CC	24370	1	50542	5
6.1	CC	28359	1	58591	6
7.1	CC	25136	1	45801	7
1.2	P	23646	2	50333	1
2.2	CC	26174	2	38227	2

……

reshape 函数比 stack 有更加复杂的句法，可在比本例单列更多列的“长形式”数据时候使用。通过设置参数 direction="wide"，reshape 函数可以完成相反任务。

2.5 平列联表

以数组的形式展示高维的列联表是相当的有难度。在定类数据分析中，这种信息通常通过包含有边的带有行列组合的因子水平对应的格子计数的二维数组来实现。这些行列典型的是“粗糙”的，只有当它们改变的时候，标签才显示出来，以明显通用的行从顶端到底端列从左端到右端的方式。在 R 中，这种平列联表可以通过 ftable 函数来实现，该函数生成一个 ftable 类，带有相应的打印方法。

以 R 标准数据的 UCBAdmissions 作为一个简单的例子，其包含一个三维的列联表，是

通过 1973 年研究生申请 UC Berkeley6 个最大的院系的人员依照录入和性别分类的情况：
data(UCBAdmissions)
ftable(UCBAdmissions)

	Dept	A	B	C	D	E	F
Admit	Gender						
Admitted	Male	512	353	120	138	53	22
	Female	89	17	202	131	94	24
Rejected	Male	313	207	205	279	138	351
	Female	19	8	391	244	299	317

这种打印形式明显比三维数组形式的要有用。

有一个 read.ftable 函数用来读取这种平列联表形式的文件。额外的参数是用来精确处理行列变量名称和测量水平的。在 read.ftable 函数的帮助中有一些有用的例子。平列联表可以从数组形式通过 as.table 函数转化维标准列联表。

请注意平列联表通过其“粗糙”的行列标签来定制。如果行的所有测量水平均被给定，可以使用 read.table 函数来读取数据，通过 xtabs 函数来生成一个列联表。

第三章： 从其他统计软件中导入

本章中，我们将讨论从其他统计软件系统中读取二进制数据文件。通常最好可以避免这种情况，然而如果原有的系统不存在，这种情况就无法避免了。

3.1 EpiInfo、Minitab、S-PLUS、SAS、SPSS、Stata、Systat

自带的 foreign 包（术语推荐包系列）提供了其他统计软件系统数据导入的工具，以及导出数据到 Stata 等。有时候，这些函数需要的内存比 read.table 函数少一些。write.foreign 函数（参见 1.2 节【导出到文本文件】，第三页）目前提供了支持导出到 SPSS 和 Stata 的机制。

EpiInfo 第五和第六版存储数据到一个自描述的固定宽度文本文件中。read.epiinof 函数可以读取'REC'格式的函数到 R 的 data.frame 类型的对象中。EpiData 也生成这样的数据文件。

函数 read.mtp 导入“Minitab Portable Worksheet”文件，返回一个工作表组成的 R 列表。

函数 read.xport 导入 SAS 的可交换格式文件，返回一个包含数据框的列表。如果 SAS 安装在电脑中，read.ssd 函数可以生成和读取 SAS 脚本，该脚本中包含了以可交换格式存储的 SAS 永久数据集。它通过 read.xport 来读取文件。包 HMisc 有一个类似的函数 sas.get。

函数 read.S 可以读取 S-PLUS3.x、4.x 和 2000 版本在 UNIX 和 Windows 系统中的二进制对象（可以在不同的操作系统中读取他们）。可以读取大部分但不是所有的 S 对象：可以读取向量、矩阵、数据框和包含这些内容的列表。

函数 data.restore 读取 S-PLUS 的转移数据（由函数 data.dump 生成），并且保存了相同

的限制（除了从 Aplpha 平台）。其还可以读取 S-PLUS5.x 和 6.x 版本的转移数据，这些数据通过 data.dump(oldStyle=T)写出。

如果以及连接到 S-PLUS，转移 S-PLUS 中的对象和在 R 中提供转移文件是可靠的。S-PLUS5.x 和 6.x 版本中需要使用 dump(……,oldStyle=T)，读取非常大的对象时，最好使用批处理脚本的转移文件，而不要使用 source 函数。

函数 read.spss 读取 SPSS 中保存和导出的文件，返回由保存数据集中每个变量组成的一个列表。SPSS 变量带有值标签的可以选择转化维 R 中的因子。

SPSS Data Entry 时生成数据输入格式的工具。缺省情况下，生成的有额外格式信息的数据文件无法被 read.spss 函数来操作，但是其可以输出数据成为普通的 SPSS 格式。

Stata 中“.dta”格式文件时二进制格式的文件。从 5，6，7/SE 和 8 版本的 Stata 的得到的文件可以通过函数 read.dta 和 write.dta 函数来读写。Stata 中的有值标签的变量可以转化维 R 中的因子。

函数 read.systat 读取在 lSystat 中 SAVE 格式的长方形数据文件（mtype=1）。这些文件以.sys 或者.syd（最近的用法）作为扩展名。

3.2 Octave

Octave 是线性代数数值运算系统，foreign 包中的 read.octave 函数可以读取 Octave 中 ASCII 数据文件的第一个向量或者矩阵，这些数据文件是 Octave 的 save -ascii 命令生成的。

第四章 ： 关系型数据库

4.1 为什么使用数据库

R 操作的数据类别是有限制的，因为 R 中所有数据都是存留在内存中的，在执行一个函数的过程中数据可能被生成好几份，R 不适应操作大数据集。超过百兆的数据对象会导致 R 耗尽内存资源。1

R 自身目前不容易支持数据获取。因为如果多个用户获取数据的时候，存在更新同一个数据，这样一个用户的操作对另外的用户就是不可见的了。

R 支持永久性数据，这样你可以把一个数据对象和一个任务的整个工作表保存，并用于随后的任务中，然而存储的数据是针对 R 的，不太容易被其他系统操作。

数据库管理系统（DBMSs），尤其是关系型数据库管理系统 2（RDBMSs）用来完成一下这些工作，其功能有如下方面：

1、 提供读取大数据集中快速选取部分数据的功能
2、 数据库中强大功能的汇总和交叉列表的功能
3、 以比长方形格子模型的电子表格 3 和 R 数据库更加严格的方式保存数据
4、 多用户并发存取数据，同时确保存取数据的安全约束
5、 作为一个服务器维大范围的用户提供服务

这样，DBMS 可能使用统计工具提取 10％的样本数据，生成交叉列表 4 从而得到一个多维

的列联表，从一个数据库中可以依照分组提取数据组进行独立的分析等等。

4.2 关系型数据库管理系统简介

传统上，有大型（并且昂贵）的商业化的关系型数据库管理系统(Informix; Oracle; Sybase; IBM's DB/2; Microsoft SQL Server on Windows)和学术的、小型系统的数据库系统(比如 MySQL, PostgreSQL, Microsoft Access 等)，前一类型极其强调数据的安全性。现在随着开源的 PostgreSQL 有了越来越高端的特点，以及“自由”版本的 Informix，Oracle 和 Sybase 在 Linux 上使用，这种界线正在变得模糊。

同时还有其他常用的数据源类型，包括电子表格，非关系型数据库，乃至文本格式文件（可能是压缩的）。开放数据库接口（ODBC）是使用这些数据源的标准。其源于 Windows (参考 http://www.microsoft.com/data/odbc/)，也可以在 Linux/Unix 上实现。

本章随后提及的包都提供了客户服务器数据库形式的服务。数据库可以驻留在本机也可以是远程（通常如此）设备上。有一个称为 SQL（结构化查询语言，读为“sequel”参见 Bowman 等，1996 和 Kline and Kline，2001）接口语言的 ISO 标准（事实上有几个：SQL92 是 ISO/IEC 9075, 也称为 ANSI X3.135-1992, 同时 SQL99 即将出台），数据库管理系统均不同程度的支持这个标准 5。

4.2.1 SQL 查询

R 中诸多接口为通常的操作生成 SQL，然而直接使用 SQL 需要复杂的操作。同城 SQL 使用大写字母拼写，但是很多用户发现在 R 接口函数中使用小写是方便的。

关系型数据库管理系统以数据库中的数据表（或者关系）作为存储数据的方式，其非常类 R 中的数据框，它们都是由相同类型的列或者叫字段的组成（数值、字符、日期、货币等等），行或者叫记录的包含整条的观测值。

SQL 查询在关系型数据库中是相当普遍的操作。下面是一些典型的 SELECT6 查询语句：

```
SELECT State, Murder FROM USArrests WHERE Rape > 30 ORDER BY Murder

SELECT t.sch, c.meanses, t.sex, t.achieve
FROM student as t, school as c WHERE t.sch = c.id

SELECT sex, COUNT(*) FROM student GROUP BY sex

SELECT sch, AVG(sestat) FROM student GROUP BY sch LIMIT 10
```

第一例是从 R 中已经被赋值到数据表中的数据框 USArrests 中选择两列，通过第三列来选择符合条件的数据，并且让结果排序。第二例完成了 student 和 school 两个数据表的连接，并且返回四列。第三里和第四例查询完成交叉表，返回频次和平均值（五个汇总函数是 COUNT，SUM, MAX, MIN 和 AVG，每个函数用于单独一列）。

SELECT 查询用 FROM 选择数据表，WHERE 确定选取条件（多个条件的时候使用 AND 或者 OR 组成），使用 ORDER BY 对结果进行排序。和数据框不同，关系型数据库中的各行

最好认为是没有顺序的，没有 ORDER BY 子句的时候，顺序是不确定的。可以对用逗号分割的多行进行排序（以字典顺序）。通过设置 DESC 在 ORDER BY 字段的后面可以反向排序。

SELECT DISTINCT 查询仅返相应数据表中回各各不同的行。

GROUP BY 子句通过准则来选择各组的行。如果指定了多列，则多维交叉分类被 5 个汇总函数进行汇总计算。HAVING 子句允许通过汇总计算得到的值来选择包含或者不好某些行。

如果 SELECT 中包含了 ORDER BY 子句会产生唯一的顺序，增加一个 LIMIT 子句可以选取（通过数值）连续一块数据的列。这对一次取回一个数据行块是有用的（当 LIMIT 子句用来优化查询的时候，除非顺序是唯一的，否则是不可靠的）。

Kline 和 Kline (2001)讨论了 SQL 在 SQL Server 2000 、Oracle、 MySQL 和 PostgreSQL 中实现的具体细节。

数据库中的数据可以保存为多种数据类型。数据类型是数据库管理系统指定的，不过 SQL 标准定义了许多类型，下列是广泛地被实现了的（通常不是通过 SQL 名称）：

float(p)	实数，可选精度，通常称为实数或者双精度数
integer	32 位整数，通常称为整数
smallint	16 位整数
character(n)	定长字符串，通常称为字符
character varying(n)	变长字符串，通常称为变长字符，有 255 字符的限制
Boolean	true 或 false，通常称为 bool 或者 bit
date	日历时间
time	当日时间
timestamp	日期时间

和时区有关，有多种时间，以及多种日期时间。文本和 bolb（表示大块文本和二进制数据）也被广泛的实现了。R 接口包广泛地的为用户实现隐藏了数据类型转换的情况。

4.3 R 接口包

在 CRAN 上有几个包可以让 R 和 DBMSs 进行通信。它们提供了不同层次的抽象。有一些提供了将整个数据框读入写出到数据库中。所有这些包中都有通过 SQL 查询语言的函数选取数据，读取结果分片（通常是不同组的行）或者整体作为数据框 7。

除了 RODBC 以外，其他所有的包都和一种 DBMS 相关，然而所有的操作都使用 DBI 包(http://developer.r-project.org/db)作为统一的前端工具和各种作为终端工具的结合起来，其中开发最好的是 RMySQL 包。在 CRAN 中终端的包还有 ROracle 和 RSQLite(和内置的数据库管理系统 SQLite 结合工作, http://www.hwaci.com/sw/sqlite)

较早的 RmSQL 和 RPgSQL 两个包当前在 CRAN 的开发领域已经终止了支持：BioConductor 项目有一个 RdbiPgSQL.PL.R (http://www.joeconway.com/plr/)，是一个把 R 嵌入到 PostgreSQL 的项目。

CRAN 上的 RMySQL 包提供了和 MySQL 数据库系统的接口（参见 http://www.mysql.com and Dubois, 2000.）。这里的表示适合 0.5-0 版：早期的版本会有本质上的差异。当前版本需要 DBI 包，这里的描述只需要做很小的改变就可以适用于其他包使用 DBI 的终端。

从 3.23.x 版本以来，在 GPL 下，MySQL 存在于 Unix/Linux 和 Windows。MySQL 是一个轻量级的数据库系统（其默认操作系统的系统文件是大小写敏感的，不同于 Windows 的情况）。RMySQL 包在 Linux 和 Windows 下均可使用。
dbDriver("MySQL")调用返回一个数据库连接管理对象，接着调用 dbConnect 函数打开数据库连接，随后可以使用泛型函数 dbDisconnect 来关闭数据库连接。相应的可以在 ROracle 或 RSQLite 中使用 dbDriver("Oracle")或 dbDriver("SQLite")
SQL 查询的发送可以通过 dbSendQuery 或 dbGetQuery 函数来实现。dbGetQuery 函数发送查询并取回结果保存为一个数据框。dbSendQuery 函数则发送查询避过那发挥一个从 DBIResult 类继承而来的对象中，这个对象可以取回结果，随后可以使用 dbClearResult 函数删除结果。函数 fetch 用来取回查询中部分或者全部的行，生成一个列表。函数 dbHasCompleted 标识是否所有的行都被取回了，dbGetRowCount 函数返回结果中的行数。
在读出、写入和删除数据库中的表方面，都有方便的接口。dbReadTable 和 dbWriteTable 函数从一个 R 的数据框中读写数据表，并把数据框的行名变为 MySQL 数据表中的 row_names 字段。

```
> library(RMySQL) # 加载 DBI 包
## 打开一个和 MySQL 的连接
> con <-dbConnect(dbDriver("MySQL"), dbname = "test")
## 列出数据库中的数据表
> dbListTables(con)
## 载入一个数据框到数据库中，删除已有同名的数据表
> data(USArrests)
> dbWriteTable(con, "arrests", USArrests, overwrite = TRUE)
TRUE
> dbListTables(con)
[1] "arrests"
## 读取整个数据表
> dbReadTable(con, "arrests")
Murder Assault UrbanPop Rape
Alabama 13.2 236 58 21.2
Alaska 10.0 263 48 44.5
Arizona 8.1 294 80 31.0
Arkansas 8.8 190 50 19.5
...
## 选择加载过的数据表
> dbGetQuery(con, paste("select row_names, Murder from arrests",
"where Rape > 30 order by Murder"))
row_names Murder
1 Colorado 7.9
2 Arizona 8.1
3 California 9.0
4 Alaska 10.0
```

```
5 New Mexico 11.4
6 Michigan 12.1
7 Nevada 12.2
8 Florida 15.4
> dbRemoveTable(con, "arrests")
> dbDisconnect(con)
```

CRAN 上的 RODBC 包给支持 ODBC8 的数据库管理系统提供了接口。它可以广泛存在，并允许相同的 R 代码和不同的数据库管理系统连通。RODBC 在 Unix/Linux 和 Windows 均可运行，并且几乎所有的数据库管理系统都支持 ODBC。我们已经测试了 Windows 上的 Microsoft SQL Server, Access, MySQL 和 PostgreSQL，以及 Linux 上的 MySQL, Oracle, PostgreSQL and SQLite 等。

ODBC 是一个客户服务器系统，可以顺利的在 Windows 客户机上连接运行在 Unix 上的数据库管理系统，反之同样可以。

在 Windows 上，通常都有对 ODBC 的支持，当前的版本可以从 http://www.microsoft.com/data/odbc/上获取，作为 MDAC 的组成部分。在 Unix/Linux 上，你也许需要一个 ODBC 驱动管理器，比如 unixODBC (http://www.unixODBC.org)或者 iOBDC (http://www.iODBC.org) 以及一个数据库管理系统的驱动程序。

在 Windows 下不仅提供了对数据库管理系统的支持，而且提供了对 Excel（‘.xls’）数据表、Dbase('.dbf')文件，甚至文本文件的支持（这些应用都不再用安装了）。

并发的连接是可能的。通过 odbcConnect 或者 odbcDriverConnect 函数调用 (Windows 界面下，允许通过对话框操作)打开一个数据库连接，返回一个随后对数据库进行操作的句柄。打印一个数据库连接可以提供 ODBC 连接的一些细节，odbcGetInfo 函数可以提供客户机和服务器的细节。

通过调用 close 或者 odbcClose 函数可以关闭一个数据库连接，也可以在一个 R 流程结束时又没有 R 的对象引用其时自动的关闭（会给出一个警告）。

一个数据库连接中各个表格的细节可以通过 sqlTables 函数得到。

函数 sqlSave 把一个 R 的数据框写入到数据库的一个表中，sqlFetch 函数把数据库中的一个表写到 R 中的一个数据框。

一个 SQL 查询可以通过 sqlQuery 函数发送到数据库中。这样返回结果到一个 R 的数据框中。（sqlCopy 发送一个查询到数据库中，并且将结果保存为数据库中的一个数据表。）

下面时一个使用 PostgreSQL 的例子，ODBC 把列和数据框名字转化为小写。我们将使用预先建好的数据框 testdb，在 unixODBC 环境下把 DSN（数据源名称）保留在'~/.odbc.ini'文件中。相同的代码可以使用 MyODBC 在 Linux 或者 Windows（这时候 MySQL 也把名字转化为小写）下针对 MySQL 使用。在 Windows 操作环境下，DSN 在“控制面板”中操作（2000/xp 版本中，“ODBC 数据源”在管理工具选项中）。

```
> library(RODBC)
## 把命名改为小写
> channel <-odbcConnect("testdb", uid="ripley", case="tolower")
## 把一个数据框写入到数据库中
> data(USArrests)
```

```
> sqlSave(channel, USArrests, rownames = "state", addPK = TRUE)
> rm(USArrests)
## 给出数据库中的数据表
> sqlTables(channel)
TABLE_QUALIFIER TABLE_OWNER TABLE_NAME TABLE_TYPE REMARKS
1 usarrests TABLE
## 展示数据库中数据表的内容
> sqlFetch(channel, "USArrests", rownames = "state")
murder assault urbanpop rape
Alabama 13.2 236 58 21.2
Alaska 10.0 263 48 44.5
...

## 一个 SQL 查询
> sqlQuery(channel, "select state, murder from USArrests where rape > 30 order by murder")
state murder
1 Colorado 7.9
2 Arizona 8.1
3 California 9.0
4 Alaska 10.0
5 New Mexico 11.4
6 Michigan 12.1
7 Nevada 12.2
8 Florida 15.4
## 移除数据库中的数据表
> sqlDrop(channel, "USArrests")
## 关闭连接
> odbcClose(channel)
```

下面是一个 Windows 下通过 ODBC 使用 Excel 电子表格的例子，我们可以通过如下方式读取电子表格：

```
> library(RODBC)
> channel <-odbcConnectExcel("bdr.xls")
## 给出电子表格
> sqlTables(channel)
TABLE_CAT TABLE_SCHEM TABLE_NAME TABLE_TYPE REMARKS
1 C:\\bdr NA Sheet1$ SYSTEM TABLE NA
2 C:\\bdr NA Sheet2$ SYSTEM TABLE NA
3 C:\\bdr NA Sheet3$ SYSTEM TABLE NA
4 C:\\bdr NA Sheet1$Print_Area TABLE NA
## 读取 sheet1 的内容
> sh1 <-sqlFetch(channel, "Sheet1")
```

```
> sh1 <-sqlQuery(channel, "select * from [Sheet1$]")
```

请注意 sqlTables 和 sqlFetch 函数数据表的不同之处，sqlFetch 可以给出差异。

ODBC 和 Excel 的接口是只读的，你不能改变电子表格中的数据。

第五章： 二进制文件

二进制连接（第六章，连接，19 页）是较好的二进制文件处理方式。

5.1 二进制数据格式

CRAN 上的 hdf5、RNetCDF 和 ncdf 包提供了和 NASA 的 HDF5 格式文件(Hierarchical Data Format, 参见 http://hdf.ncsa.uiuc.edu/HDF5/)的接口，以及和 UCAR 的 netCDF 数据文件(network Common Data Form, 参见 http://www.unidata.ucar.edu/packages/netcdf/)的接口。

上述两种文件格式都是用来存储数组方式组织的科学技术数据的文件系统，其中包括描述、标签、格式、单位等。HDF5 还提供了各组数组，R 接口使得列表和 HDF5 的分组数据对应起来，可以读写数值和字符型的向量和矩阵。

CRAN 上的 ncvar 包通过使用 RNetCDF 的功能提供了高层次的和 netCDF 的接口。

还有来自于 http://www.bioconductor.org 上的 rhdf5 包。

5.2 dBase files (DBF)

Ashton-Tate 发明的 dBase 是 DOS 下的程序，后来为 Borland 公司拥有，它带有".dbf"扩展名，作为一种二进制“平”文件格式变得相当流行。它已经被'Xbase'系列的数据库系统使用了，包括 dBase、Clipper、FoxPro 和他们相应的 Windows 版本 Visual dBase、Visual Objects 和 Visual FoxPro (参见 http://www.e-bachmann.dk/docs/xbase.htm)。一个 dBase 文件中包含一个头，随后是一系列的字段，非常类似于 R 的数据框。数据本身以文本格式存储，可以包含字符、逻辑和数值型字段，后续的版本中还有其他类型(参见 http://clicketyclick.dk/docs/data_types.html)。

在所有的平台中，R 都提供了 read.dbf 和 write.dbf 函数读写基础 DBF 文件。在 Windows 环境下，可以使用 RODBC 包中的 odbcConnectDbase 函数，它提供了通过 Microsoft's dBase ODBC 驱动程序读取 DBF 文件的全面功能(Visual FoxPro 驱动也可以通过 odbcDriverConnect 函数实现)。

第六章： 连接

R 中提供的连接依照了 Chambers 在 1998 年提出的意见，这个系列的函数通过使用灵活的接口连接到类似文件的对象而不是使用文件名。

6.1 连接类型

最常见的连接类型是一个文件连接，R 中文件连接使用 file 函数生成。文件连接（如果操作系统允许的话）可以打开一个文件以便于以文本或二进制的格式读、写或者增加文件。事实上，打开的文件既可以读也可以写，R 为读写保存一个独立的文件位置。

请注意，在缺省情况下当一个连接生成时它还没有被打开。规则就是说如果连接没有打开，一个函数使用它之前必须要打开它（必须的），如果打开连接后，使用以后关闭连接。简而言之，保持你发现的连接的状态。通用函数 open 和 close 将显式的打开和关闭连接。通过函数 gzip 提供的算法压缩后的文件可以作为由函数 gzfile 产生的连接来使用，同时通过函数 bizp2 压缩的文件可以通过 bzfile 函数使用。

Unix 下的程序员习惯针对特殊文件使用 stdin、stdout 和 stderr 函数。它们作为 R 中终端连接而存在。它们也可能时正常的文件，但是可以也用于在 GUI 控制台输入和输出。（即便是使用 Unix 下标准的 R 接口，stdin 函数引用从 readline 函数生成的行结果而不是使用文件。）

这三个终端连接常常是开放的，不能被打开或者关闭。函数 stdout 和 stderr 通常用于正常的输出和错误信息展示。他们可以使用在相同的场合，不过正常的输出可以使用函数 sink 反向读入，输出至 stderr 函数的错误使用 sink 函数的时候，必须设置参数 type=”message”。请注意此处的用语：连接不能是逆向的，但是输出可以发送到其他连接。

文本连接是输入的另外一个来源。使得 R 的字符向量可以像从文本文件中逐行读取。调用函数 textConnection 生成并打开一个连接，连接生成时该函数复制当前字符串向量到一个内部的缓冲区。

文本连接也可以用于获取 R 输出到一个字符串向量。函数 textConnection 可以生成一个新的字符对象或者添加内容到一个已有的字符对象上，两种情况都发生在用户的工作间。调用 textConnection 函数的时候连接打开，完整的列输出到 R 对象的已有连接中。关闭连接写出任意剩余的输出到字符串向量的最后一个元素中。

Pipe 是一种连接到其他过程的特殊文件形式，pipe 连接使用函数 pipe 来生成。打开一个供写入的 pipe 连接（或者是追加内容到 pipe）运行操作系统命令，连接其标准输入到 R，随后将内容写入到连接。逆过程是打开一个输入的 pipe 连接，运行操作系统命令，从连接中获得 R 中已有的标准输出。

“http://”、”ftp://”、”file://”等 URL 类型可以使用函数 url 读取。为了方便起见，在调用 url 函数时，可以将文件作为 file 参数来指定。

在大多数支持 socket 的 Unix、Linux 和 Windows 操作系统中，Socket 可以通过函数 socketConnection 来作为连接使用。Socket 可以读写，客户端和服务端 socket 都可以使用。

6.2 输出到连接

我们已经说明了 cat、write、write.table 和 sink 作为文件写入的函数，可以通过设置参数 append=TRUE 在文件后追加内容，而且这是 1.2.0 版本以前的优先做法。现在的情形类似，当 file 参数是字符串，文件连接是打开的（供写入或者追加内容），在调用函数后再关闭。如果我们想反复的写出内容到同一个文件，显式的声明并打开一个连接，把连接对象作为输出函数的参数调用会更加有效率。这些也可以用来写入到 pipe 中，早期通过 file="|cmd” 来实现这些功能（现在也在使用）。

函数 writeLines 把完整的文本行写出到一个连接中。下面是一些简单的例子：

```
zz <-file("ex.data", "w") # 打开一个输出文件连接
```

```
cat("TITLE extra line", "2 3 5 7", "", "11 13 17",
file = zz, sep = "\n")
cat("One more line\n", file = zz)
close(zz)
## Unix 环境下，使用 pipe 在输出中把数字间隔符号转化为逗号
## R 字符串和 shell 中需要双反斜杠
zz <-pipe(paste("sed s/\\\\./,/ >", "outfile"), "w")
cat(format(round(rnorm(100), 4)), sep = "\n", file = zz)
close(zz)
## 现在看输出文件：
file.show("outfile", delete.file = TRUE)
## 使用 lm 函数帮助中的例子，获取 R 输出
zz <-textConnection("ex.lm.out", "w")
sink(zz)
example(lm, prompt.echo = "> ")
sink()
close(zz)
## 现在 'ex.lm.out' 中包含了供进一步处理的输出内容
## 如下方式，查看其中内容
cat(ex.lm.out, sep = "\n")
```

6.3 从连接中输入

从连接中读取内容的基本函数是 scan 和 readLines。将一个字符串作为参数，打开一个文件连接作为函数的持久调用，显式的打开一个文件连接可以以不同的格式连续的读取内容。

调用了 scan 函数的其他函数也可以使用连接，特别的是 read.table 函数，下面的一些简单的例子：

```
## 读取上个例子中生成的文件
readLines("ex.data")
unlink("ex.data")
## 读取当前目录下的文件列表(Unix 环境下)
readLines(pipe("ls -1"))
# 假设一个“data”文件中包含如下内容
450, 390, 467, 654, 30, 542, 334, 432, 421,
357, 497, 493, 550, 549, 467, 575, 578, 342,
446, 547, 534, 495, 979, 479
# 读取如上内容
scan(pipe("sed -e s/,$// data"), sep=",")
```

为了方便性，如果 file 参数指定了一个 FTP 或者 HTTP URL，URL 通过 url 函数来读取。通过 file://foo.bar 指定文件也是可以的。

6.3.1 Pushback

C 程序员可能比较熟悉 ungetc 函数，用来在文本输入中对一个字符进行压栈。R 连接已一种更强大的方法实现相同的想法，函数 pushBack 可以把任意数量的文本行压栈到一个连接上。Pushback 的操作类似于堆栈，读取要求首先使用最近被压栈的文本中的每一行，随后是其先的压栈，最后是连接本身内容的读取。一旦被压栈行完整读取后，其就被清除了。压栈的数量可以通过函数 pushBackLength 来设置。

一个简单的例子展示了这个想法：

```
> zz <-textConnection(LETTERS)
> readLines(zz, 2)
[1] "A" "B"
> scan(zz, "", 4)
Read 4 items
[1] "C" "D" "E" "F"
> pushBack(c("aa", "bb"), zz)
> scan(zz, "", 4)
Read 4 items
[1] "aa" "bb" "G" "H"
> close(zz)
```

压栈仅供文本模式输入下的连接使用。

6.4 列出和操作连接

用户当前打开的所有连接可以通过 showConnections 函数列出，使用参数 all=TRUE 可以把已经关闭或者中止的连接展示出来。

泛型函数 seek 可以用来读取数据，以及在一些连接中重置读写的当前点。不幸的是，该函数依赖于操作系统的功能，这些可能是不可信的（比如 Windows 下的文本文件）。函数 isSeekabel 函数通过输入参数连接对象可以报告连接的位置能否改变。

函数 truncate 可以在当前位置截断一个已打开的供写入的文件。

6.5 二进制连接

函数 readBin 和 writeBin 读写二进制连接。以二进制模式打开一个连接，使用“b”来设置这种模式，“rb”表示读取，“wb”或者“ab”用于写入。函数参数等使用情况如下：

readBin(con, what, n = 1, size = NA, endian = .Platform$endian)

writeBin(object, con, size = NA, endian = .Platform$endian)

每个 con 是一个连接，在调用过程中需要的时候打开，如果是使用一个字符串，那么就默认为是指定一个文件名。

写入的情况是类似的。对象可以是原子型的向量对象，是没有定义属性的数值型、整数型、逻辑型、字符型、复数型或者 raw 型向量。默认情况下，像在内存中那样，作为一个精确的比特流写出到文件，

函数 readBin 从文件中读取比特流，通过 what 设置其模式，作为一个向量读取。设置

可以是一个合适模式的对象或者一个描述模式的字符串。参数 n 定义了从连接中读取向量的元素的最大个数。参数 signed 允许 1-byte 或者 2-byte 的整数作为符号整数（默认情况）或者无符号整数。

其他两个参数用于与其他程序和操作平台进行数据读写交换。默认情况下，二进制数据直接在内存和连接之间交换。为了和其他程序交换数据，可能需要做更多的设置，比如读取 16-bit 的整数，或者写入单精度的实数。这可以使用 size 参数来设置，整形和逻辑型变量常常可以设置为 1、2、4、6，实数型常常可以设置为 4、8、12 或者 16。不同存储大小间转换会损失一些精度，包含 NA 的向量中不能如此。

字符串以 C 语言格式读写，也就是说每个字符串后面使用表示 0 的比特数作为结尾。函数 readChar 和 writeChar 提供了更多的灵活性。

6.5.1 特殊值

函数 readBin 和 writeBin 可以通过缺失值和特定的值，虽然大小发生变化不是期望的情况。

R 中的逻辑值和整数值类型的缺失值是 INT_MIN，C 语言“limits.h”中定义的 int 型可以表达的最小的整数，通常对应于比特模式的“0xffffffff”。

R 中数值型和复数型的特定值和机器相关，有时候和编译器有关。使用它们最简单的办法是在连接到外部的应用中去除 Rmath 库，这个库输出 NA_REAL, R_PosInf 和 R_NegInf，并且在头文件“Rmath.h”中给出了 ISNAN 和 R_FINITE 的宏定义。字符缺失值写为 NA。

第七章：网络接口

在底层水平在网络连接之间交换数据方面，R 提供了一些有限的功能。

7.1 从 socket 中读取

R 基本包提供了一些功能，它们使得支持 BSD socket 的系统之间实现通信（包括常见的 Linux、Unix 和 Windows 下 R 的 port）。使用 socket 方面功能的潜在的问题在于为安全原因或者使用了 Web caches 而受到阻，因此这些函数在内部网比在外部网更加有用。对一个新项目而言，socket 连接可以使用。

早期底层接口通过 make.socket、read.socket、write.socket 和 close.socket 等函数来实现。

7.2 使用 download.file 函数

download.file 函数提供了通过 FTP 和 HTTP 方式从网页上读写文件的功能。通常都不会使用这个函数，因为 read.table 和 scan 这些函数可以直接从 URL 上读取数据，或者显式的使用 url 打开一个连接，或者隐式的把 URL 作为 file 参数的值。

7.3 DCOM 接口

DCOM 是不同程序 Windows 下的通信协议，并可能在不同的机子上实现。CRAN Software->Other->Non-standard 目录下由 Thomas Baier 创造的 StatConnector 程序提供了 DLL 的代理接口，和 Windows 版本下的 R 连接，生成一个 DCOM 服务端。这样可以从 R 交换简单的对象（向量和矩阵），并执行到 R 的命令。

这个程序还有一个 Visual Basic 的例子以及 Erich Neuwirth 的一个 Excel 插件。这个接口和这儿讨论的是相反方向的情况，其他程序（Excel 或者 Visual Basic 等）成为客户端，而 R 作为服务端。

另外一个(D)COM 服务端可以从 http://www.omegahat.org/上得到，它允许 R 对象导出为一个 COM 对象值。这个网页上还有 RDCOMClient 和 SWinTypeLibs 包，允许 R 作为(D)COM 的客户端。

7.4 CORBA 接口

CORBA 和 DCOM 类似，允许应用程序调用其他程序中服务端对象的方法和操作，而且可能是不同语言编写的代码，还运行在不同的机器上。在 Omegahat 项目(在 http://www.omegahat.org/RSCORBA/)中有一个 CORBA 包，目前是为 Unix 环境提供的，Windows 版本也可能出来。

这个包可以让 R 命令找到存在的 CORBA 服务端，查询它们提供的方法，动态的激活这些对象的方法。R 作为参数在这些调用中读入，使得操作进行。基础的数据类型（向量和列表）可以缺省的进入，复杂的对象通过引用载入。例子中包含了和 Gnumeric (http://www.gnumeric.org)电子表格配合使用的情况，也包含了和数据可视化系统 ggobi 配合使用的例子。

也可以在 R 中创建 CORBA 服务端，允许其他应用程序调用这些方法。比如说，可以提供对某一个数据集的连接，或者到 R 的建模软件的连接。这是通过动态的组合 R 的数据对象和函数完成的。这样可以显式的从 R 导出数据和函数。

也可使用 CORBA 包在 R 上完成平行分布式的计算。一个 R 程序段作为管理者，发送任务到 R 中运行的其他程序段。这样可以在 R 中产生异时或后台的 CORBA 调用。更多的信息请参考 http://www.omegahat.org/RSCORBA/上的 Omegahat 项目。

第八章 读取 Excel 电子表格

最常见的 R 数据导入导出问题要数“我怎样读取 Excel 电子表格”了。本章汇总了一下上面关于这个问题的建议和选择。

首先的建议是除非必要，最好不要如此做。如果你已经连接到 Excel 了，那么以制表符分割或者逗点分割的方式导出 Excel 中你想要的数据，使用 read.delim 或者 read.csv 函数读取数据到 R 中（在使用逗号作为小数点的欧洲大陆地区需要使用 read.delim2 或者 read.csv2 函数）。如果你没有 Excel，在 Windows 和 Unix 下都有很多查那古训可以读取电子表格格式的数据文件并且导出为文本格式，比如说 Gnumeric (http://www.gnome.org/projects/gnumeric/) 和 OpenOffice (http://www.openoffice.org)。你也可以使用剪切复制的方法从这些程序中展示

出来的电子表格和 R 交换数据：read.table 函数可以从 R 的控制台读取数据，在 Windows 下，可以通过剪贴板实现（通过设置参数 file="clipboard"或者 readClipboard 函数）。

值得注意的是，一个 Excel 的".xls"文件中包含不仅一张电子表格：而是包含了多张电子表格，而且表格中还包含了公式、宏等等。并非所有的读入都可以完成第一张表以外的表格，也许会被文件中的其他内容弄糊涂。

Windows 下的用户可以通过使用 RODBC 包中的 odbcConnectExcel 函数来操作。它可以选择 Excel 电子表格中的任何一张表格的某些行列。

Perl 使用者发布了一个叫 OLE::SpreadSheet::ParseExcel 的组件，可以通过 xls2csv.pl 程序将 Excel 电子表格转化为 CSV 格式文件。gdata 包通过 read.xls 函数提供了一个包装好的基本功能。

附录 A：参考文献

1) R. A. Becker, J. M. Chambers and A. R. Wilks (1988) The New S Language. A Programming Environment for Data Analysis and Graphics. Wadsworth & Brooks/Cole.

2) J. Bowman, S. Emberson and M. Darnovsky (1996) The Practical SQL Handbook. Using Structured Query Language. Addison-Wesley.

3) J. M. Chambers (1998) Programming with Data. A Guide to the S Language. Springer-Verlag. P. Dubois (2000) MySQL. New Riders.

4) M. Henning and S. Vinoski (1999) Advanced CORBA Programming with C++. Addison-Wesley.

5) K. Kline and D. Kline (2001) SQL in a Nutshell. O'Reilly.

6) B. Momjian (2000) PostgreSQL: Introduction and Concepts. Addison-Wesley. Also downloadable at http://www.postgresql.org/docs/awbook.html.

7) T. M. Therneau and P. M. Grambsch (2000) Modeling Survival Data. Extending the Cox Model. Springer-Verlag.

8) E. J. Yarger, G. Reese and T. King (1999) MySQL & mSQL. O'Reilly.