HZ BOOKS
华章科技

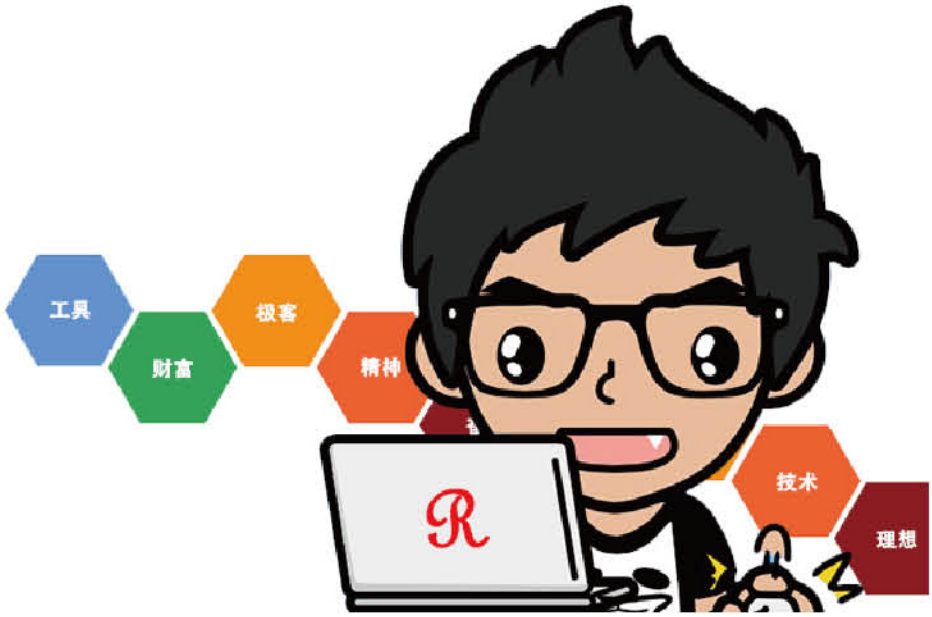
资深R语言用户多年实战经验的结晶，介绍了各种性能奇特的R语言包，提升R语言性能的方法，以及R语言在实际使用时与Java、MySQL、MongoDB、Hive、HBase、Hadoop等技术的综合运用的解决方案。

数据分析
技术丛书

R的极客理想

工具篇

张丹/著



机械工业出版社
China Machine Press

数据分析技术丛书

R的极客理想——工具篇

张丹 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

R 的极客理想——工具篇 / 张丹著 . —北京：机械工业出版社，2014.8
（数据分析技术丛书）

ISBN 978-7-111-47507-1

I. R… II. 张… III. 程序语言 – 程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2014）第 170133 号

本书首先介绍了 R 的工具包、时间序列包和性能监控包；然后阐述 R 语言与其他编程语言的通信，以及 R 语言作为服务器的应用；最后阐释 R 语言与各种数据库的通信以及 R 语言如何与 Hadoop 集成。附录介绍了 Java、各种数据库以及 Hadoop 的安装方式。书中内容涉及计算机、互联网、数据库、大数据、统计、金融等领域，详细总结了 R 语言在实际使用时与 Java、MySQL、Redis、MongoDB、Cassandra、Hadoop、Hive、HBase 等技术综合运用的解决方案，具有实战性及可操作性强等特点。

本书适合所有 R 语言工作者，包括软件工程师、DBA、数据科学家、科研工作者以及相关专业的学生。读者可以选择任何感兴趣的章节进行阅读，每节之间没有特别的顺序要求。

R 的极客理想——工具篇

张丹 著

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：明永玲 责任校对：董纪丽
印 刷： 版 次：2014 年 8 月第 1 版第 1 次印刷
开 本：186mm×240mm 1/16 印 张：19.5
书 号：ISBN 978-7-111-47507-1 定 价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

Foreword 序　一

生活中我是三分熟的牛排，张丹兄是五分熟的牛排，一天他给我发邮件邀请我写序，我好些天都没答复，为什么呢？因为我们互相都不熟啊！嘿嘿，好冷。从没有人找我写过序，张丹有胆找一个浑身负能量爆棚的新手及话痨写，还说自由发挥，风格不限，作者自己都不怕，我怕啥，大不了把脖子往你们面前一横！当然那是张丹的脖子……其实我自己看书一般不太看推荐序（除非里面有重大八卦），因为推荐序里通常也就是友情帮点一百二十八个赞，你要是像鲁迅先生那样半夜爬起来翻看推荐序，当然你肯定不会看见“吃人”两个字，而是看见一个特殊的变量值：NULL。哎呀，我这只是夸张的说法啦，推荐序没那么糟糕，我也并非和张丹完全不熟，我看过他写的很多博客文章。好了，言归正传。在下谢益辉，身患统计学永久性脑损伤，目前在 RStudio 当码农，天天蹲 A 村村口敲代码。十年前在中国人民大学统计学院上大三（哥也年轻过），侥幸没被统计计算课上的 R 语言折磨死，按照我们疯人院（没有含沙射影的意思，请读者切勿自动匹配）的规矩：你上回没弄死我，这次你在前面跑，换我来弄死你好了。就这样 R 语言成了我快乐生活的一部分。咦，为什么有一种细思恐极的感觉？

作为一名不那么极的 R 极客，我自然乐意看到一本写给极客的书。在这个数据时代（千万别再跟我提“大”数据，否则我立刻变身纯生牛排给你看），各种新技术风起云涌，需要有人坐一段时间冷板凳，为我们收拾整理这些技术，让那些从证明三种中心极限定理的苦海中逃出来的研究生们毕了业不要立马又陷入另一片五种数据库引擎的苦海。我花了一晚上加一白天的时间才看完《 R 的极客理想》的书稿，看不懂和看得懂的地方都跳着看，看不懂主要是因为我没有计算机背景（我学习的第一门计算机语言是 VB，你们笑我可以，别笑出声就好，不过我还写过 VBScript 呢，这次你们可以笑出声了），例如 Java，我多年前曾经自学过一阵，现在都忘光了，写个“你好世界”的程序都需要搜一下；看得懂的地方主要是纯 R 的内容，不过有些地方还是慢慢看了，比如 formatR 包那里，主要是看张丹有没有黑我。算他识相没黑我，那好，我在序言里可以放心……黑他了；江湖险恶，你不中

箭谁中箭？

整本书涵盖的内容比较广，每一节的篇幅相对较短，我觉得这种写作风格挺好，每天晚上睡前打开一节看看，在口水浸湿书之前应该可以看完一节，既能学到知识又合理利用了睡前时间。如作者自己所说，这书不是写给初学者的，所以看第 1 章的不要被“ R 基础”的标题给骗了，全是些奇门遁甲之术，一个基础包都没介绍！虽然我不敢讲 R 是不是最值得学习的程序语言，但我对 1.1 节的观点深表同意：R 往往用两三行代码解决问题，不会时时考虑最优与否（很多需要优化的地方都已经用 C/Fortan 等底层语言打包好了）。R“它爹”S 语言的主要作者 John Chambers 在几十年前就说了：S 语言的目的是让我们（统计学家）快速而可靠地把脑子里的想法变成软件。将数据拿到手之后甩开膀子从各个角度去分析就好了，想画图画图，想跑模型跑模型，而不必先考虑定义一个结构体以及某个成员是整型还是浮点型。书里主要用到的操作系统是 Windows 和 Ubuntu，不过苹果（OS X）用户不必担心，R 在 OS X 上跑起来也是妥妥的，个人推荐用 Homebrew 安装 R。第 1 章介绍了一系列奇特 R 包，从中我可以看出作者钻研这些包的乐趣，极客需要这种小乐趣的推动，后面书中我们还会持续看到作者钻研的迹象，他胆敢把自己失败的经历都写出来，实在是很勇敢，但这也反映了极客的真实生活嘛，哪有干什么都一帆风顺的？奇特 R 包系列里包括了我的 formatR 包，这个包有几处小小的坑，提醒广大读者注意看文档，代码格式主要还是得靠程序员自觉，不能老依靠 formatR 这样的自动整理代码的包。fortunes 包里的确有大智慧（不是炒股软件），但我感觉主要还是各种恶搞和冷笑话，码农的心思你别猜，别猜别猜。RStudio 服务器版是个好东西，未来的趋势可能是啥都存储在服务器上，浏览器控制一切，再也没有黑乎乎的 SSH 窗口。不过前两天我一个同学给我打电话哭诉我坑了他，因为 RStudio 的服务器版太好用了，他把代码都放服务器上，每天就在浏览器里写代码，不再从本地传来传去，结果服务器硬盘坏了，没备份。我感到很不好意思，这周打羽毛球我都没敢再叫他。RJSONIO 的作者本来是我很敬佩的一位大极客，可能是 R 界兴趣最广泛的极客之一，可他的代码极少写单元测试。导致我被坑了几次之后越来越不敢相信他的编码质量，哎，上天给聪明极客的惩罚就是让他们失去单元测试。不过话说回来，什么 JavaScript/JSON 啊，HTML/CSS 啊，都是非常容易入门的技术，作为统计出身的我，大力推荐大家掌握一点这种低投入高产出而且又有趣的计算机知识。Cairo 是另一位大极客的作品，我在他手下呆过两个多月，其聪明程度以及掌握的计算机知识之广总令我惊叹，例如后面章节中介绍的 Rserve、FastRWeb 以及 RCassandra 都是他单枪匹马的作品，一个比一个极，问题就是极过了以至于很少有人知道，所以也没见广泛应用。有时候我都想，你懂得这么多东西你家里人知道吗？他家里人知不知道没关系，起码现在我们知道张丹帮了个大忙，让他的几份作品至少多了一些中国读者。有志于成为极客的码农们，在忍受孤独的同时，我觉得真的应该好好想想为什么你做的东西没有广为人知，就算你不想这个问题，也该想想怎么

到今天还没有女朋友吧，这两个问题背后肯定有共同原因。如张丹引用我的话所说，Cairo 包的高质量在现今的 R 版本中已经不算太大优势了，因为 R 本身已经支持 Cairo 库，例如其 PNG 图片输出和 Cairo 包的输出质量上几乎是一样的，但有一个特例除外，就是散点图中点的形状为某一个类型的圆点时（看不懂这句话的人请查阅 points 函数的帮助），基础 R 画出来有锯齿，而 Cairo 包没有，至于 pch 取值多少时有锯齿，这问题就留给你们自己去探索吧。caTools 这个神奇到莫名其妙的包，某种程度上反映了一些 R 包作者广泛的兴趣以及没有正规计算机背景的特点，这事情难说好坏。

也许是缘于对金融的兴趣，张丹在第 2 章介绍了时间序列数据的处理。金融是才疏学浅的我不太能理解的行业（注定了我穷困一生的命运），时间序列也是我比较弱的功课，除了 R 自带的 ts() 函数以及简单的 ARMA 模型，我脑子里剩下的时间序列的知识已经没多少了。zoo 和 xts 是我听过无数遍但从来没使过的 R 包，从此也可以看出 R 的应用领域太广泛了，我用了 10 年 R 也没用过这两个流行的包，主要是因为我不太做时间序列相关的工作。看完这一章我觉得读者不妨也研究一下 R 的基础图形，主要是 graphics 包，掌握一点基本的画点画线技术，对图形的灵活应用会很有帮助，如今 R 的图形天下基本被 ggplot2 占据，但我还是老土的基础“图形党”（没办法，我学 R 的时候 ggplot2 还没出生），我觉得不是所有的数据都适合 ggplot2 的。这一章介绍了时间序列数据处理以及可视化，如果是做时间序列的预测，我听说过无数遍 forcast 包，但至今未得一试。

R 的性能一直以来都是计算机专业人士对 R 的槽点，张丹和我都表示没有压力，但这不表示性能不是问题，当然谁都希望自己的程序跑得快，好省下几分钟时间去写另外 300 行 C 代码提升下一个程序的速度。第 3 章提供了提速以及找代码瓶颈的工具。对于 memoise 包本身我其实没太大兴趣，但它的源代码是值得一看的，R 里面的函数（或称闭包，Closure）和环境是很有意思的话题，路远坑多，慢走不送。性能监控也是一个优化代码的重要手段，作者介绍了基础工具 Rprof() 以及酷炫工具 lineprof 包，让我们知道自己的代码的瓶颈所在。最后作者讲，R 语言需要更多 IT 人的推动，我实在不能同意更多。R 作为统计学家写给统计学家的语言，总是会有些坑，需要专业人士来帮忙填补；另一方面，这帮顽固的统计学家完全无视界面的重要性，看看 R 官方网站有多朴素就知道了，简直是土得连渣都掉不下来，你跑去跟他反映，他只是一道冰冷眼神就可以杀死你。话说做网页前端的 IT 人士你们在哪儿呢？

第二部分的两章介绍 R 的服务器应用，前面说了我是 Java 外行，所以不敢乱点评。只记得那时候 Urbanek 大人一路上手舞足蹈给我解释 FastRWeb 的原理，然而我回去看到 /var/FastRWeb/ 这个目录的时候就已经暗自决定抛弃它了。我个人对那些需要 sudo 才能运行且存放在非标准位置的程序有抵触心理，因为我下次一定会忘记怎么运行它以及配置文件在什么位置。Rserve 和 FastRWeb 在它们被发明的年代里绝对都是划时代的，在服务器

端跑基于 R 的服务是很多码农的梦想，还记得那些年我们一起追的 Rweb 吗？如今回头看看，还有多少人记得并用着 CGI？可喜的是，张丹在第 5 章也介绍了 WebSocket 技术，这也是一个相对较新、很有趣且有用的话题，建议读者好好研究。书中提到 websockets 包已经被移出 CRAN，什么原因我也不知道，不过我基本上确定 httpuv 包可以取代它，也许是 websockets 的作者看到 httpuv 的工作之后决定不要重复劳动了。httpuv 包是 shiny 包的核心技术之一，如今捣鼓 R 的服务器端应用，怎么能忽略 shiny？ shiny 比 Rserve/FastRWeb 出现晚了近十年，为什么前者迅速流行起来，而后者尽管带着划时代的思想和技术，却被广大用户忽略，极客们应该再次好好想想。有读者可能会说，切，shiny 是你们厂（RStudio）的产品，你当然自卖自夸啦！是不是这样呢，我们且留给时间检验。

第三部分的两章就是数据库八仙过海各显神通了，我仅仅粗浅了解一点 MySQL，请 Hadoop/NoSQL 同行们不要笑出声。作为入门教程，这些章节都不错，从安装到“你好世界”的例子都有介绍，十八般武艺入门之后的事情大家都知道，遇到问题搜索就是了。

极客不是一种身份，而是一种态度。在我眼里，这个词是中性的，极客不代表一个人有多牛，而是他的钻研态度、好奇心以及对新技术的识别和接受能力。有些很牛很聪明的人，未必能把聪明才智转化为生产力（请勿对号入座）。张丹这本书给大家提供了一条通向 R 的极客之路，但这绝对不是终点。技术人士容易沉迷于技术，就像科学研究人士迷信某一种科学一样，唉，我就是这样浑身负能量。希望读者通过这本书能感受到作者探索的乐趣，保持开放心态，积极学习，然后找到适合自己的极客理想（以及女朋友！相信我，后者会让前者更快实现）。写序似乎应该说点鼓励的话吧，我没写过也不清楚贵圈的规矩，那么就引用麦太的话好了：从前有一位小朋友他很努力学习，后来他发财了。

谢益辉

2014 年 6 月 23 日于 A 村

（吝啬的房东一直不给我修空调，已热哭，决定在最后这个黄金广告位狠狠黑她一把，叫她随便得罪码农！）

Foreword 序　二

我有时会问自己会不会因为名字而买一本书？当我看到“R 的极客理想”时，我就找到了答案。把这个书名做个分词，去除停止词之后有三个词：“R”、“极客”和“理想”。这是耐人寻味的三个词。

R 是我现在谋生的工具，我对它有着十多年的感情。我亲历了 R 从和 GAUSS 进行比较的时代走过和 SAS 进行比较的时代、来到了和 Python 及 Julia 这样的语言进行比较的时代，从最开始的无人问津到如今的炙手可热。R 的资料在互联网上可以说汗牛充栋，但是中文书籍仍然很少。张丹是圈内著名的博主，明永玲是圈内著名的编辑，有幸被邀约写这篇序言时，我对这个组合非常看好。

阅读内容之后，我发现这本书有太多和其他 R 语言书籍不一样的地方。传统的 R 语言书籍大多是基于统计的思维展开的，通过介绍统计方法在 R 中的实现来学习 R，这就使得很多统计出身的用户可以很容易地和其他统计软件进行类比，从而加速学习的过程。进入大数据的时代后，R 作为数据处理的神器也越来越受关注，R 语言的书籍也开始以数据为中心，从数据的获取、处理、分析一直到可重复研究和可视化展现，在应用层面进行全方位的介绍。但是，作为一种编程语言，程序员视角的 R 书籍还是非常少的。张丹的这本书刚好可以覆盖这部分的内容。

极客是 R 圈中比较少见的一种生物，尤其是来自极客之乡 IT 界的正宗极客。R 的最初用户基本上都是统计圈的，但是最近几年 R 能够在国内越来越火，主要得益于 IT 界的贡献。R 在欧美火的时间更早，但发展的趋势也大抵如此。从 R 的本性来说，它本不该是极客关注的语言，因为其对外部功能延伸的追求远甚于对内部语言完美的追求，从 S 语言设计之初就声明了人的时间远比计算机的时间宝贵，尤其是分析建模人员的时间。这种比较乡愿的风格最初是不为极客所喜的。但是因为其极端易用，在惜时如金的产业界快速地流行了起来，自然而然地产生了大量的难题。于是高贵冷艳、魅惑狂狷的极客们就参与进来了。我认识的张丹，就是这样一位极客。

R 是一个很奇怪的东西，没有编程基础的人可以很容易地入门，但是很难有信心觉得自己成了高手。编程高手初学时常常是破口大骂，但是很快就中了 R 的迷毒。说到底，还是因为 R 本质上是统计学家发明的语言，和模型打交道的能力比和计算机打交道的能力更强大。虽然如此，随着 R 的扩展包越来越丰富，用户在享受便利性的同时也增加了理解上的风险，函数背后的机制不再是黑箱，那么，R 中的 IT 高手的见解就变得非常重要。从他们的视角来看 R、使用 R，无论是对于统计背景还是 IT 背景的用户都有很强的借鉴意义。

R 流的是实用主义的血，看上去和理想是背道而驰的。但 R 毕竟是没有灵魂的工具，它的性格应该取决于用它的人。在张丹的这本书里，我看到了理想的光辉。我看过不少书也自己写书，感觉介绍知识是最简单的事，但是表明观点是最困难的。对于作者来说，观点越不鲜明就越能避免犯错。但是对于读者来说，尤其是初学的读者，很容易陷入对自己的怀疑中。而张丹的这本书会直接告诉读者应该怎样做，照着代码操作一遍就能解决问题。虽然有些建议可能不是最好的解决方案，但至少是足够好的，在实际的应用中可以解决问题。在这一点上，理想主义的作者和实用主义的 R 实现了完美的结合。

当然，整本书在“R”、“极客”、“理想”之间实现了更加完美的结合。

李舰

2014 年 8 月 1 日于上海

Preface 前　言

为什么要写这本书

我是一名程序员，前后做了 10 年的程序开发工作。在这 10 年间，我从程序员一路做到架构师，经历了太多的系统和应用。我做过手机游戏，写过编程工具；做过大型 Web 应用系统，写过公司内部 CRM；做过 SOA 的系统集成，写过基于 Hadoop 的大数据工具；做过外包，做过电商，做过团购，做过支付，做过 SNS，也做过移动 SNS。以前只用 Java，然后开始用 PHP……如同其他程序员一样，我一度陶醉于追求各种技术的创新，但始终有一个问题困扰着我，那就是如何才能够将我所掌握的技术转变成价值？这就好比我面对着一座金山，我拥有先进的技术，可以制作各种性能稳定、功能卓越的挖掘机器，但我不懂如何将矿石提纯，变成金子！每每看到别人利用我的技术挖掘出金子时，我只能满脸的羡慕，心中无限的不甘。

直到遇见 R 语言，我豁然开朗。R 语言为我从另外一个角度开启了宝藏的大门，也让我对自己的职业重新思考、规划，最后坚定了我向统计、金融行业的转型。如果你也存在以上的问题，不如随着本书一起进入 R 语言的世界，领略 R 语言特有的魅力，通过对 R 语言的学习，重新认识大数据的价值，更深一步地提升个人价值。

随着我与统计、金融领域的朋友交流地逐步深入，我深刻地体会到，他们对 R 语言的实际使用也存在着很大的问题和困惑。比如，他们在某些实验室环境下，使用 R 语言可以很轻松、很顺利地实现预期效果，但是移植到真实环境下，面对浩瀚繁复的大数据，在使用 R 语言的过程中出现了很多问题。这就好比面对一座金山，他们掌握着先进的提纯技术，但他们所使用的挖掘、采集工具却还停留在石器时代！使用工具的落后，使他们要面对大量 R 语言之外的问题，这让他们应接不暇，甚至崩溃！有的人甚至因此认为，R 语言只是一种实验室语言，至少以现在的技术水平无法将它运用到现实生活中，R 语言在现实生活中广泛应用，简直是天方夜谭！

是的，如果你是一名没有计算机背景的 R 语言使用者，你在实际使用中也同样会遇到

许多这样或那样的问题，面对这些棘手的问题寝食难安，尝试着通过各种方式寻求解决方案。其实，在计算机领域，这些问题已经早就有了成熟、有效的解决方案。

本书的内容来自我在 R 语言实际使用过程中的经验总结，基本都是我在工作中使用 R 语言的真实记录，其中涉及计算机、互联网、数据库、大数据、统计、金融等领域，详细总结了 R 语言在实际使用时与 Java、MySQL、Redis、MongoDB、Cassandra、Hadoop、Hive、HBase 等技术综合运用的解决方案，具有实战性，可操作性强。如果你与 R 语言接触时间不长，本书可以让你看到 R 语言在各行业、各领域所散发的魅力；如果你在某行业使用 R 语言已经有一段时间了，可能在使用 R 语言的过程中遇到了瓶颈，本书将让你看到 R 语言在与其他计算机语言结合后所迸发的强大活力；如果你是技术人员，本书中有全局观的案例实施，也许会给你带来新的启发，甚至跟我一样，重新规划自己的职业生涯，找到学习、奋斗的新方向；如果你是企业的中高层管理者，在本书中可以看到我们已经实现的技术成果，如果需要，你甚至可以按照书中记录的详细操作步骤，直接在企业环境中实施，直接获利！

在此，我不得不强调，本书不是入门书，不讲 R 的语法，如果你想学习 R 语言的基础语言入门知识，那么，你来错地方了。但是，如果你已经具备了一定的 R 语言基础，但不一定具有计算机语言背景，我将告诉你 R 语言在真实环境下到底都能够做什么，并且详细地告诉你怎样一步一步地实施。

在与各界 R 语言初学者的交流中，我发现，入门后，学习 R 语言最大的问题，在于如何使用 R 语言的众多软件包，而介绍这方面的图书很难找到，只有一些网上流传的小册子。本书涉及了 30 个 R 语言包，并结合我的使用心得及案例分析，相信会解决大家 R 语言入门后的困扰。

本书是“ R 的极客理想”系列图书的第一本，姊妹篇《 R 的极客理想——高级开发篇》将深入介绍 R 语言底层原理，并使用 R 语言开发出企业级的应用。

本书的使用环境涉及 Linux Ubuntu 和 Windows 7 两种操作系统，R 语言包的 2.15.3 和 3.0.1 两个版本，在每一节中都有明确的标识。

R 语言还在不断地进步和更新，它将引导一场数据的革命，跨学科的结合是时代趋势，也是我们的机遇！

读者对象

本书适合以下 R 语言工作者：

- ❑ 计算机背景的软件工程师；
- ❑ 数据库背景的 DBA；
- ❑ 数据分析背景的数据科学家；
- ❑ 统计背景的科研工作者；
- ❑ 大专院校相关专业的学生。

如何阅读本书

本书的内容分为四个部分。

第一部分是 R 基础（第 1 ~ 3 章），介绍了为什么要学习 R 语言，R 语言不同版本的安装，以及 R 语言中常用的 12 个软件包。帮助读者快速了解 R 语言的工具包、时间序列包和性能监控包。

第二部分是 R 服务器 (第 4 ~ 5 章) ，介绍了 R 语言与其他编程语言的通信，以及 R 语言作为服务器的应用。帮助读者打通 R 语言与其他编程语言的通道，并实现 R 语言的服务器应用。

第三部分是数据库和大数据 (第 6 ~ 7 章)，介绍了 R 语言与各种数据库的通信，以及 R 语言与 Hadoop 集成。帮助读者打通 R 语言与各种数据库层的通道，并实现 R 语言对基于 Hadoop 大数据的处理。

第四部分是附录，介绍了 Java、各种数据库以及 Hadoop 的安装方式。笔者希望读者可以在不借助其他参考书的情况下，完成书中所有实例。

本书为工具书，每节之间没有特别的顺序要求，你可以选择任何你感兴趣的章节进行阅读。如果你是一名初学者，想全面掌握 R 语言，请按顺序阅读全部的章节。

勘误和支持

由于笔者的水平有限，加之编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。为此，笔者创建一个在线的图书交流网站 (https://onbook.me)，方便与读者进行沟通。如果读者在阅读过程中遇到问题，也可以在网站中留言，我将尽量在线上为你提供最满意的解答。书中的全部源代码都可以从华章公司网站（www.hzbook.com）或本书交流网站下载，我也会及时更新代码。本书为黑白印刷，关于更绚丽的彩色图片，读者运行源代码即可看到。如果你有什么宝贵意见，欢迎发送邮件至 bsspirit@gmail.com，期待能够得到你真挚的反馈。

致谢

感谢我的团队，林伟林、林伟平、邓一硕，让我们因 R 语言走到一起。感谢机械工业出版社华章公司的编辑明永玲，帮助我审阅全部章节，引导我顺利完成书稿。感谢我的爸爸、妈妈和爱人，感谢你们对我工作上的支持和生活上的照顾！

谨以此书献给我最亲爱的家人以及众多 R 语言爱好者们！

张丹

2014 年 5 月于中国北京

目　　录 *Contents*

序一
序二
前言

第一部分　R 基础

第 1 章　R 语言基础包 …… 2
1.1　R 是最值得学习的编程语言 …… 2
1.1.1　我的编程背景 …… 3
1.1.2　为什么我会选择 R …… 3
1.1.3　R 的应用前景 …… 7
1.1.4　时代赋予 R 的任务 …… 8
1.2　R 的历史版本安装 …… 8
1.2.1　R 在 Windows 中安装 …… 9
1.2.2　R 在 Linux Ubuntu 中安装 …… 10
1.2.3　R 的最新版本安装 …… 10
1.2.4　R 的指定版本安装 …… 10
1.3　fortunes 记录 R 语言的大智慧 …… 11
1.3.1　fortunes 介绍 …… 12
1.3.2　fortunes 安装 …… 12
1.3.3　fortunes 包的使用 …… 12
1.4　formatR 代码自动化排版 …… 13

1.4.1 formatR 介绍 …… 13
1.4.2 formatR 安装 …… 14
1.4.3 formatR 的使用 …… 14
1.4.4 formatR 的源代码解析 …… 20
1.4.5 源代码中的 Bug …… 21
1.5 多人在线协作 R 开发 RStudio Server …… 22
1.5.1 RStudio 和 RStudio Server …… 22
1.5.2 RStudio Server 安装 …… 22
1.5.3 RStudio Server 使用 …… 23
1.5.4 RStudio Server 多人协作 …… 26
1.6 R 和 JSON 的傻瓜式编程 …… 29
1.6.1 rjson 包介绍 …… 29
1.6.2 RJSONIO 包介绍 …… 33
1.6.3 自定义 JSON 的实现 …… 36
1.6.4 JSON 性能比较 …… 38
1.7 R 语言的高质量图形渲染库 Cairo …… 40
1.7.1 Cairo 介绍 …… 40
1.7.2 Cairo 包安装 …… 40
1.7.3 Cairo 使用 …… 41
1.8 caTools：一个奇特的工具集 …… 46
1.8.1 caTools 介绍 …… 47
1.8.2 caTools 安装 …… 48
1.8.3 caTools 使用 …… 48

第 2 章 时间序列基础包 …… 58

2.1 R 语言时间序列基础库 zoo …… 58
2.1.1 zoo 包介绍 …… 59
2.1.2 zoo 安装 …… 60
2.1.3 zoo 包的使用 …… 60
2.2 可扩展的时间序列 xts …… 75
2.2.1 xts 介绍 …… 75
2.2.2 xts 包的安装 …… 78
2.2.3 xts 包的使用 …… 78

2.3 时间序列可视化 plot.xts ····· 93
2.3.1 xtsExtra 介绍 ····· 93
2.3.2 xtsExtra 安装 ····· 93
2.3.3 xtsExtra 包的使用 ····· 94

第 3 章 R 性能监控包 ····· 104

3.1 R 语言本地缓存工具 memoise ····· 104
3.1.1 memoise 介绍 ····· 105
3.1.2 memoise 安装 ····· 105
3.1.3 memoise 使用 ····· 105
3.1.4 memoise() 函数源代码分析 ····· 106
3.2 R 语言性能监控工具 Rprof ····· 108
3.2.1 Rprof() 函数介绍 ····· 109
3.2.2 Rprof() 函数的定义 ····· 109
3.2.3 Rprof() 函数使用 : 股票数据分析案例 ····· 109
3.2.4 Rprof() 函数使用 : 数据下载案例 ····· 112
3.2.5 用 profr 包可视化性能指标 ····· 113
3.2.6 Rprof 的命令行使用 ····· 115
3.3 R 语言性能可视化工具 lineprof ····· 116
3.3.1 lineprof 介绍 ····· 117
3.3.2 lineprof 安装 ····· 117
3.3.3 lineprof 使用 ····· 118

第二部分 R 服务器

第 4 章 R 语言的跨平台通信 ····· 122

4.1 Rserve 与 Java 的跨平台通信 ····· 122
4.1.1 Rserve 安装 ····· 123
4.1.2 用 Java 远程连接 Rserve ····· 124
4.2 Rsession 让 Java 调用 R 更简单 ····· 126
4.2.1 Rsession 下载 ····· 126
4.2.2 用 Eclipse 构建 Rsession 项目 ····· 127

4.2.3 Rsession 的 API 介绍 ……128
4.2.4 Rsession 使用 ……129
4.3 解惑 rJava R 与 Java 的高速通道 ……132
4.3.1 rJava 介绍 ……133
4.3.2 rJava 安装 ……133
4.3.3 rJava 实现 R 调用 Java ……134
4.3.4 rJava(JRI) 实现 Java 调用 R (Windows 7) ……135
4.3.5 rJava(JRI) 实现 Java 调用 R (Ubuntu) ……137
4.4 Node.js 与 R 跨平台通信 ……137
4.4.1 Node.js 简单介绍 ……138
4.4.2 R 语言配置环境 ……138
4.4.3 Node.js 配置环境 ……139
4.4.4 Node.js 与 R 跨平台通信 ……139

第 5 章 R 的服务器实现 ……143

5.1 R 语言服务器程序 Rserve 详解 ……143
5.1.1 Rserve 的启动 ……144
5.1.2 Rserve 高级使用：Rserve 配置管理 ……146
5.1.3 Rserve 高级使用：用户登录认证 ……148
5.2 Rserve 的 R 语言客户端 RSclient ……149
5.2.1 配置 Rserve 服务器 ……150
5.2.2 RSclient 安装 ……150
5.2.3 RSclient 的 API ……151
5.2.4 RSclient 的使用 ……152
5.2.5 两个客户端同时访问 ……152
5.3 FastRWeb：跑在 Web 上的 R 程序 ……153
5.3.1 FastRWeb 介绍 ……154
5.3.2 FastRWeb 安装 ……155
5.3.3 FastRWeb 使用 ……156
5.4 R 语言构建 Websocket 服务器 ……159
5.4.1 websockets 介绍 ……159
5.4.2 websockets 安装 ……160
5.4.3 快速启动 websockets 服务器 demo ……162

5.4.4 R 语言创建 Websocket 服务器实例……163
5.4.5 R 语言创建 Websocket 客户端连接……163
5.4.6 用浏览器 HTML5 原生 API 客户端连接……164

第三部分 数据库和大数据

第 6 章 数据库和 NoSQL……168

6.1 RMySQL 数据库编程指南……168
6.1.1 RMySQL 在 Linux 下安装……169
6.1.2 RMySQL 在 Windows 7 下安装……173
6.1.3 RMySQL 函数使用……176
6.1.4 RMySQL 案例实践……181
6.2 R 利剑 NoSQL 之 MongoDB……183
6.2.1 MongoDB 环境准备……183
6.2.2 rmongodb 函数库……185
6.2.3 rmongodb 基本使用操作……187
6.2.4 rmongodb 性能测试的案例……189
6.3 R 利剑 NoSQL 之 Redis……192
6.3.1 Redis 环境准备……192
6.3.2 rredis 函数库……193
6.3.3 rredis 基本使用操作……194
6.3.4 rredis 测试案例……198
6.4 R 利剑 NoSQL 之 Cassandra……200
6.4.1 Cassandra 环境准备……200
6.4.2 RCassandra 函数库……201
6.4.3 RCassandra 基本使用操作……202
6.4.4 RCassandra 使用案例……204
6.4.5 Cassandra 的没落……205
6.5 R 利剑 NoSQL 之 Hive……206
6.5.1 Hive 环境准备……207
6.5.2 RHive 安装……208
6.5.3 RHive 函数库……209

6.5.4 RHive 基本使用操作 ……209
6.6 用 RHive 从历史数据中提取逆回购信息 ……212
6.6.1 逆回购简介 ……212
6.6.2 历史数据存储结构 ……213
6.6.3 通过用 RHive 提取数据 ……213
6.6.4 策略模型及实现 ……216

第 7 章 RHadoop ……222

7.1 R 语言为 Hadoop 注入统计血脉 ……222
7.1.1 Hadoop 介绍 ……223
7.1.2 为什么要让 Hadoop 结合 R 语言 ……224
7.1.3 如何让 Hadoop 结合 R 语言 ……225
7.1.4 展望未来 ……226
7.2 RHadoop 安装与使用 ……226
7.2.1 环境准备 ……227
7.2.2 RHadoop 安装 ……227
7.2.3 RHadoop 程序开发 ……229
7.3 RHadoop 实验：统计邮箱出现次数 ……233
7.3.1 需求描述 ……233
7.3.2 算法实现 ……234
7.4 RHadoop 实现基于 MapReduce 的协同过滤算法 ……236
7.4.1 基于物品推荐的协同过滤算法介绍 ……236
7.4.2 R 语言本地程序实现 ……237
7.4.3 R 基于 Hadoop 分步式程序实现 ……242
7.5 rhbase 安装与使用 ……249
7.5.1 HBase 环境准备 ……250
7.5.2 rhbase 安装 ……250
7.5.3 rhbase 函数库 ……251
7.6 解决 RHadoop 安装错误：PipeMapRed.waitOutputThreads() ……253
7.6.1 rmr2 运行错误日志 ……254
7.6.2 定位错误到 Hadoop 日志 ……255
7.6.3 从 Hadoop 入手找解决办法 — 失败 ……256
7.6.4 从 RHadoop 入手找解决办法 — 成功 ……257

第四部分　附　　录

附录 A　Java 环境安装 ……262

附录 B　MySQL 数据库安装 ……266

附录 C　Redis 数据库安装 ……270

附录 D　MongoDB 数据库安装 ……273

附录 E　Cassandra 数据库安装 ……277

附录 F　Hadoop 安装 ……280

附录 G　Hive 环境安装 ……287

附录 H　HBase 安装 ……290

第一部分 Part 1

R 基础

- 第1章　R语言基础包
- 第2章　时间序列基础包
- 第3章　R性能监控包

Chapter 1 第 1 章

R 语言基础包

本章主要介绍了为什么要学习 R 语言，R 语言软件的安装，R 语言的开发工具，以及 R 语言中常用的几个软件包，以帮助读者快速了解 R 语言的工具包，激发读者对 R 语言的学习兴趣。

1.1 R 是最值得学习的编程语言

问题

为什么要学 R 语言？

引言

如果要问在 Node、Lua、Python、Ruby 和 R 这 5 种语言中，哪个语言在 2014 年的应用前景会更好，我会毫不犹豫地选择 R，而且我认为 R 语言不仅是 2014 年，也是以后更长一段时间内的明星。在本书开篇，我们就谈谈为什么 R 语言是最值得学习的编程语言。

1.1.1　我的编程背景

本人是程序员、架构师，从编程入门到今天，一直深信着 Java 是改变世界的语言，Java 已经做到了，而且一直很辉煌。但当 Java 越来越强大，覆盖领域越来越多，变得无所不能的时候，它反而不够专业，这就给了其他语言发展的机会。

我已使用 Java 语言 11 年，R 语言 3 年，Node 1 年，对于这个问题“哪个语言在 2014 年的应用前景会更好”，我选择 R 语言。

1.1.2　为什么我会选择 R

从下面的 9 个方面来说明我选择 R 的原因。

- ❑ R 的基因
- ❑ R 的发展
- ❑ R 的社区和资源
- ❑ R 的哲学
- ❑ R 的使用者
- ❑ R 的语法
- ❑ R 的思维模式
- ❑ R 解决的问题
- ❑ R 的不足

1. R 的基因

1992 年，新西兰奥克兰大学的 Ross Ihaka 和 Robert Gentleman 两位统计学家，为了方便教授初等统计课程，发明了一种编程语言，因为他们名字的首字母都是 R，于是 R 便成为这门语言的名称。

从开始学习 R 语言，我就开始了知识的跨界思考。统计学基于概率论，概率论又基于数学，用计算机的方式编程，同时解决某个领域的实际问题。多种学科知识的交集，决定着我们解决问题的能力。统计的基因，让 R 语言与众不同！

2. R 的发展

R 一直在小众领域成长着，最早也只有统计学家在用，主要用来代替 SAS 做统计计算。然而时代在进步，随着大数据的爆发，R 终于在这一波浪潮中被工业界所发现。然后，越来越多有工程背景的人加入到这个圈子，对 R 的计算引擎、性能以及各种程序包进行改进和升级，让 R 获得了新生。

我们现在用到的 R 语言软件，已经越来越接近工业软件的标准了。由工程师推动的 R 的发展，其速度远远地超过了由统计学家推动的发展。随着人们对数据分析需求的进一步增加，R 会以更快的速度继续发展，将成为免费的、开源的数据分析软件的代名词。

3. R 的社区和资源

R 的发展离不开 R 的各种社区支持，尤其是 R 的官方社区支持。在 R 的官方网站中，我们可以下载到 R 语言软件、R 的第三方软件包和 R 的其他支持软件。当然，我不得不承认 R 的官方网站 (http://www.r-project.org/) 从 Web 页上看起来太简陋了，稍微调整一下 CSS 样式表，都会比现在好看很多。也许这种简单、无修饰也是统计学家的基因吧。R 语言的社区资源同其他语言一样丰富，除了官方社区，还有开发者论坛 (http://r.789695.n4.nabble.com/)、R-Journal 列表 (http://journal.r-project.org/)、软件包列表、R 语言图书列表以及 R 用户组等。

R 是自由软件，因此开发者可以开发自己的软件包，封装自己的功能，然后在 CRAN (http://cran.rstudio.com/) 上面发布。截止到 2014 年 2 月，共有 5236 个 R 包在 CRAN 上面发布。可能很多人会说只有 5236 个包，数量太少了。这是因为 CRAN 是需要提交申请的，每个包都必须经过 R 语言小组审核、检查后才会发布出来，而且审核非常严格。高质量是发布一个新的 R 包的基本要求。由于 CRAN 过于严格的审查，让很多开发者选择在 RForge(https://r-forge.r-project.org/) 上发布 R 包，还有些 R 包是基于 Github 发布的，我也在 Github 上面发布了自己的 R 包：https://github.com/bsspirit/chinaWeather 。

下面列出与 R 语言相关的主要社区和资源。

- ❑ R 官方网站：http://www.r-project.org/
- ❑ R 开发者论坛：http://r.789695.n4.nabble.com/
- ❑ CRAN：http://cran.rstudio.com/
- ❑ RForge：https://r-forge.r-project.org/
- ❑ R 新闻和博客：http://www.r-bloggers.com/
- ❑ 统计之都：http://cos.name/

4. R 的哲学

每种语言都有自己的设计理念和哲学，而我体会的 R 的哲学就是“静下心做事情”。R

不需要很长的代码，也不需要设计模式。一个函数调用，传几个参数，就能实现一个复杂的统计模型。我们需要思考的是用什么模型、传什么参数，而不是怎么进行程序设计。我们可能会用 R 实现“从一个数学公式，变成一个统计模型”的过程，我们也可能会考虑“如何让一个分类器结果更准确”，但我们不必思考一个算法的“时间复杂度是多少，空间复杂度是多少”。

R 的哲学，可以让你把数学和统计学的知识，变成计算模型，这也是 R 的基因所决定的。

5. R 的使用者

R 语言早期主要是学术界统计学家在用，后来逐渐被其他很多领域的学者所用。R 语言有各种不同的应用领域，包括统计分析、应用数学、计量经济、金融分析、财经分析、人文科学、数据挖掘、人工智能、生物信息学、生物制药、全球地理科学、数据可视化等。

近几年，由互联网引发的大数据革命让工业界的人开始认识 R，加入 R。当越来越多的有工程背景的人加入到 R 语言使用者的队伍后，R 才开始向着全领域发展，逐步实现工业化的要求。现在，R 已不仅仅是学术界的语言，它还是工业界必备的语言。

下面列出一些推动 R 语言在工业界发展的 R 包。

- ❑ RevolutionAnalytics 公司的 RHadoop 产品，让 R 可以直接调用 Hadoop 集群资源。
- ❑ RStudio 公司的 RStudio 产品，给了我们对编辑软件新的认识。
- ❑ RMySQL、ROracle、RJDBC 打通了 R 和数据库之间的访问通道。
- ❑ rmongodb、rredis、RHive、rHBase、RCassandra 打通了 R 和 NoSQL 数据库之间的访问通道。
- ❑ Rmpi、snow 打通了单机多核并行计算的通道。
- ❑ Rserve、rwebsocket 打通了 R 语言的跨平台通信的通道。

6. R 的语法

R 是面向对象语言，语法如同 Python。但 R 的语法很自由，很多函数的名字看起来都是那么随意，这也是 R 的哲学的一部分吧！例如，看到如下这样的赋值语法，有其他语言基础的程序员，肯定会崩溃的。

```
> a<-c(1,2,3,4)->b
> a
[1] 1 2 3 4
> b
[1] 1 2 3 4
```

随机取正态分布 N(0,1) 的 10 个数，又是这么的简单。

```
> rnorm(10)
 [1] -0.694541401  1.877780959 -0.178608091  0.004362026
 [5]  0.836891967  1.794961298  0.115284187  0.155175219
 [9]  0.464028612 -0.842569561
```

用 R 画鸢尾花的数据集的散点图，有非常好的可视化效果。

```
> data(iris) #加载数据集
> head(iris) #查看前6行数据集
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

> plot(iris) #画图
```

输出结果见图 1-1。

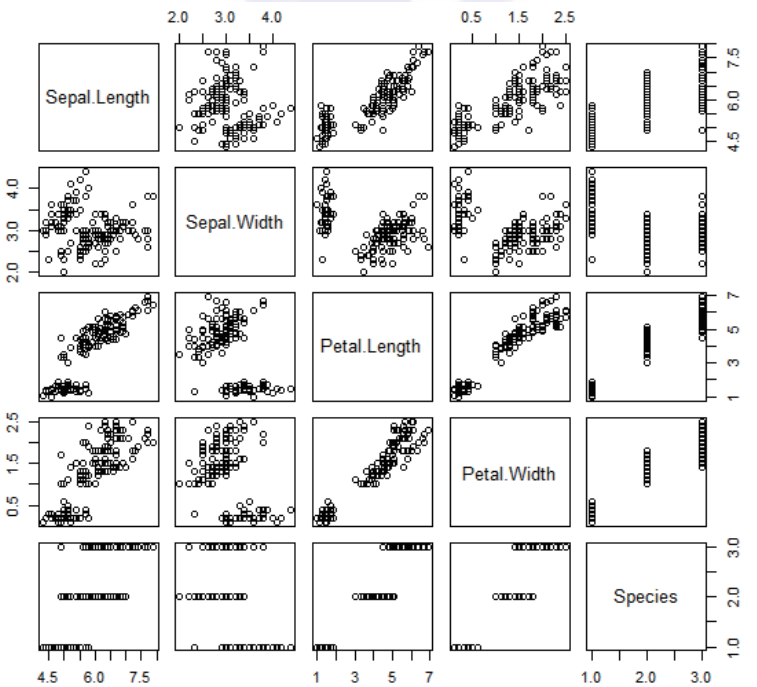


图 1-1 鸢尾花的数据集的散点图

正是因为 R 自由的哲学，让 R 的语法独特而简洁，我已经喜欢上这种哲学了。

7. R 的思维模式

R 语言让我跳出了原有的思维定式。使用 R 语言，我们应该像统计学家那样思考问题，而不是拘泥于程序员的思维模式。统计学家的思维模式，是先考虑为什么，再考虑做什么。而程序员的思维模式，是直接考虑怎么做，等有了结果再考虑为什么。

R 语言是直接面向数据的语言。在我们的日常生活中，无论做什么事情都会产生数据，上网有浏览数据，买东西有消费数据，就算什么都不干，也会受大气 PM2.5 的影响，有空气污染指数数据。利用 R 语言，我可以直接分析这些数据。面向什么业务，就分析什么数据，不需要从产品经理向程序员的角色转换，不需要考虑有什么功能，更不需要考虑程序设计的事。跳出程序员的思维模式，我们所能认知的东西会更多，于是也能找到更适合自己的定位。

8. R 解决的问题

当数据成为生产资料的时候，R 就是为人们能运用生产资料创造价值的生产工具，R 语言主要解决的是数据的问题。整个人类文明所获得的全部数据中，有 90% 以上是自互联网诞生以来产生的；当 Hadoop 帮助人们解决了大数据存储的问题后，如何发现数据的价值，则成为当前最火的话题。R 语言具有强大的统计分析能力，这就让它成为数据分析最好的工具。所以，R 要解决的问题，就是如何挖掘数据价值的问题。

9. R 的不足

尽管前面说了 R 的各种优点，但我们依然不能说 R 就是完美无缺的，因为 R 也有很多不足。具体来说，R 的缺点有下面 5 个。

- R 软件是统计学家编写的，并不如软件工程师编写的软件那么健壮。
- R 软件的性能，存在一些问题。
- R 语言很自由，语法命名不太规范，需要花时间熟悉。
- R 语言的内核编程，要比普通的 R 包使用，难度大得多。
- R 语言结合了很多数学、概率、统计的基础知识，学起来有一定门槛。

R 的这些不足，都是可以克服的。当有更多有工程背景的人加入的时候，R 语言会比现在更强大，会帮助使用者创造更多的价值。

1.1.3　R 的应用前景

R 可以做所有 SAS 能做的事情。SAS 系统全称为 Statistics Analysis System，是国际上最知名的商业分析软件工具之一。SAS 用于决策支持的大型集成信息系统，其重要组成部分和核心功能是统计分析功能。在数据处理和统计分析领域，SAS 系统被誉为国际上的标

准软件系统，堪称统计软件界的巨无霸。

R 和 SAS 处于完全的竞争的关系中，R 的免费和开放，让 R 有着更广阔的应用前景。下面给出当今 R 应用最热门的领域。

- ❑ 统计分析：统计分布、假设检验、统计建模。
- ❑ 金融分析：量化策略、投资组合、风险控制、时间序列、波动率。
- ❑ 数据挖掘：数据挖掘算法、数据建模、机器学习。
- ❑ 互联网：推荐系统、消费预测、社交网络。
- ❑ 生物信息学：DNA 分析、物种分析。
- ❑ 生物制药：生存分析、制药过程管理。
- ❑ 全球地理科学：天气、气候、遥感数据。
- ❑ 数据可视化：静态图、可交互的动态图、社交图、地图、热图、与各种 JavaScript 库的集成。

本书会介绍 R 语言在统计分析、金融分析、数据挖掘、推荐系统、社交网络等领域的应用。R 有着非常广阔的应用前景，而且 R 也将成为新一代的最有能力创造价值的工具。

1.1.4 时代赋予 R 的任务

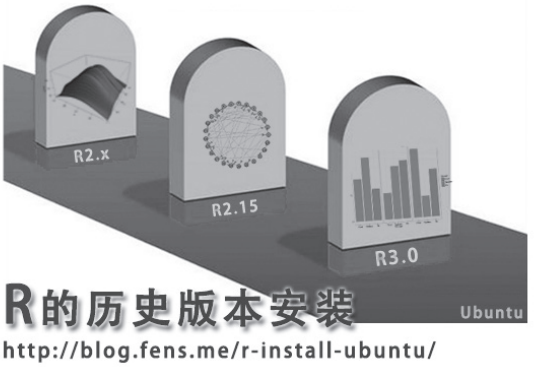
R 语言是在大数据时代被工业界了解和认识的语言，R 语言被时代赋予了挖掘数据价值、发现数据规律以及创造数据财富的任务。R 语言也是帮助人们发挥智慧和创造力的最好的生产工具，因此我们不仅要学好 R 语言，还要用好 R 语言，为社会注入更多的创新的生产力。

总而言之，在这 5 种语言中，R 是最特殊的，R 被赋予了与其他语言不同的使命。R 的基因决定了 R 将成为 2014 年，也可能是以后更长一段时间的明星。因此我认为“R 是最值得学习的编程语言”。不论你正在读书，还是已经工作，掌握 R 语言这个工具并找最适合自己的位置将会前途无量。

1.2 R 的历史版本安装

问题

在 Linux Ubuntu 上，如何安装不同版本的 R？



引言

R 语言已进入到了 3.0 的时代，但有些第三方的 R 包还处于 2.15 的状态，没有升级，如 RHadoop 等。我们要用这些 R 包的时候，就需要指定版本的 R 软件。对于 Windows 来说，这是很简单的操作，只要安装不同的 (.exe) 文件就行了；对于 Linux 系统来说，就不那么容易了，需要我们手动进行配置。不熟悉 Linux 系统的同学，在这里就很容易卡住。所以，本节就讲一下如何在 Linux Ubuntu 系统中安装 R 语言软件包的指定版本。

1.2.1　R 在 Windows 中安装

通过 R 的官方网站 (http://cran.r-project.org/)，我们可以下载 Linux、MacOS、Windows 系统的 R 语言安装包。R 在 Windows 系统中安装非常简单，下载可执行文件 (.exe)，双击即可进行安装。安装后就能运行 R 语言的界面，如图 1-2 所示。

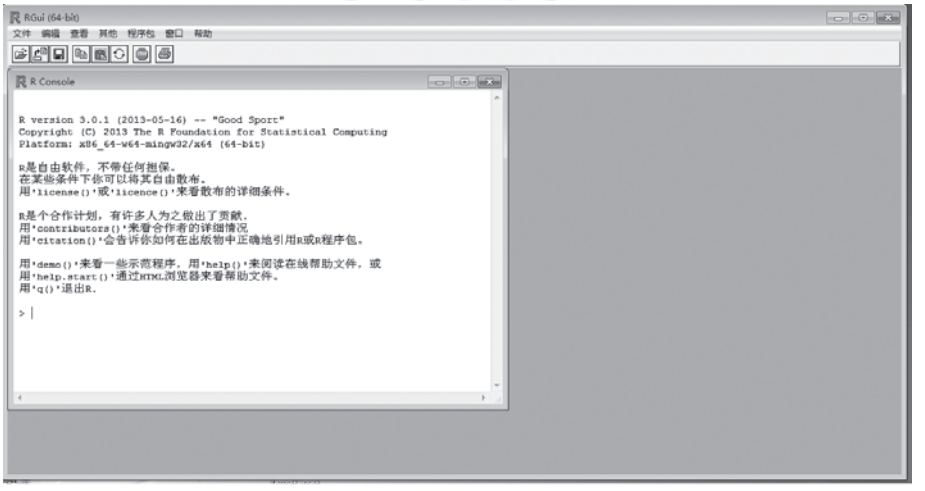


图 1-2　R 在 Windows 系统中的安装界面

1.2.2 R在Linux Ubuntu中安装

本书使用的Linux是Ubuntu 12.04.2 LTS 64bit的系统，安装R语言软件包可以通过Ubuntu的apt-get工具进行安装。下面就介绍在Linux Ubuntu中安装R语言的过程。

```
~  R  #检查R是否已安装
The program 'R' is currently not installed.  You can install it by typing:
sudo apt-get install r-base-core

~ sudo apt-get install r-base-core  # 根据提示安装R语言软件包
~ R --version  # 检查R的版本
R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-linux-gnu (64-bit)
```

前面的检查结果表明，我们安装的是R的默认版本，即2.14.1版，这与本书中R的版本是不符的，接下来我们希望安装最新版本R的软件包。

1.2.3 R的最新版本安装

首先，删除Linux Ubuntu系统中原有的R软件包，代码如下：

```
~ sudo apt-get autoremove r-base-core  # 删除系统中原有的R软件包
```

接下来，找到一个Ubuntu的软件源镜像（http://mirror.bjtu.edu.cn/cran/bin/linux/ubuntu/ ），Linux Ubuntu 12.04对应的名字是precise，进入到precise/目录，找到r-base-core相关的文件，发现有多个R的版本。把这个软件源，增加到apt的sources.list文件中，代码如下：

```
  ~ sudo sh -c "echo deb http://mirror.bjtu.edu.cn/cran/bin/linux/ubuntu
precise/ >>/etc/apt/sources.list"  # 在sources.list文件最下面，新加一行

~ sudo apt-get update  # 更新源
~ sudo apt-get install r-base-core  # 再次安装R语言软件包
~ R –version  # 检查R的版本
R version 3.0.3 (2014-03-06) -- "Warm Puppy"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
```

这时我们就安装了最新的R语言版本——3.0.3版。

1.2.4 R的指定版本安装

由于本书中的例子，大部分是基于3.0.1完成的，而RHadoop的例子是基于2.15.3完

成的，因此我们还需要指定 R 的安装版本。

1. 安装 R 的 2.15.3 版本

```
~ sudo apt-get autoremove r-base-core  # 删除系统中原有的R软件包
~ sudo apt-get install r-base-core=2.15.3-1precise0precise1  # 安装R的2.15.3版本
~ R –version   # 检查R语言软件包版本
R version 2.15.3 (2013-03-01) -- "Security Blanket"
Copyright (C) 2013 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-pc-linux-gnu (64-bit)
```

2. 安装 R 的 3.0.1 版本

```
~ sudo apt-get autoremove r-base-core  # 删除系统中原有的R软件包
~ sudo apt-get install r-base-core=3.0.1-6precise0  # 安装R的3.0.1版本
~ R –version   # 检查R语言软件包版本
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
```

这样我们就可以很方便地指定安装不同版本的 R 的语言软件包，满足不同的应用需求！

1.3　fortunes 记录 R 语言的大智慧

问题

如何才能更深入地了解 R，它的起源、成长、经历是怎样的？

引言

R语言是在大数据“火”起来后，映入人们眼球的。但R语言的社区已经存在很多年，我们并不知道，R语言在很长的历史时期，有着什么样的智慧。不过，正有人悄悄地记录着R语言的大智慧。

1.3.1 fortunes介绍

fortunes库是一个R语言的语录集，截至2013年12月14日，一共总结了360条R-help的留言。这些都是R语言智慧的精华，让R语言的后辈使用者，可以更了解R语言的本身，了解R的精神。

1.3.2 fortunes安装

本节使用的系统环境是：

- Linux: Ubuntu 12.04.2 LTS 64bit
- R: 3.0.1 x86_64-pc-linux-gnu

注 fortunes同时支持Windows 7环境和Linux环境。

Fortunes的安装过程如下。

```
~ R  # 启动R程序
> install.packages("fortunes")  # 安装fortunes包
> library(fortunes)  # 加载fortunes包
> ?fortunes  #查看帮助
```

1.3.3 fortunes包的使用

fortunes包的使用非常简单，只有一个函数fortune()。

```
> fortune()  # 随机查看一条语录
Barry Rowlingson: Your grid above has 8*6 = 42 points.
(That was a subtle Hitchhikers Guide To The Galaxy reference there, honest, and
not a stupid dumb multiplication mistake on my part after working four 18-hour
days on the trot...)
Peter Dalgaard: [...] Don't panic, just throw yourself at the ground and miss.
   -- Barry Rowlingson and Peter Dalgaard
      R-help (March 2004)

> fortune(108)  # 指定查看一条语录
Actually, I see it as part of my job to inflict R on people who are perfectly
happy to have never heard of it. Happiness doesn't equal proficient and
```

```
efficient. In some cases the proficiency of a person serves a greater good than
their momentary happiness.
   -- Patrick Burns
      R-help (April 2005)
```

完整的语录下载地址是 http://cran.r-project.org/web/packages/fortunes/vignettes/fortunes.pdf。静下心来阅读这些智慧精华就能更了解 R 语言本身。想用好一门语言，就需要更深入地了解它。

1.4　formatR 代码自动化排版

问题

如何写出让别人看得懂，且符合规范的代码呢？

引言

新手写的代码，大都不注重代码规范，以为实现功能就算完成了。这种代码不仅别人不愿意读，过几个月后再看自己都会觉得很烂。不仅仅新手如此，大多数程序员写的代码都没有考虑如何让别人看着更方便。程序员最痛苦的事情，不是每天加班写程序，而是每天加班读懂别人写的程序。最后，有人实在忍受不了其他人的丑陋代码，便开始制定代码编程规范，又有人去实现代码的自动化排版工具。formatR 就是这样的一个 R 语言自动化排版的工具。

1.4.1　formatR 介绍

formatR 包是一个实用的包，提供了 R 代码格式化功能，可以自动设置空格、缩进、换

行等代码格式，让代码看起来更友好。formatR 包中的 API 中主要有下面 5 个函数。

- ❑ tidy.source: 对代码进行格式化
- ❑ tidy.eval: 输出格式化后的 R 代码和运行结果
- ❑ usage: 格式化函数定义，并按指定宽度输出
- ❑ tidy.gui: 一个 GUI 工具，支持编辑并格式化 R 代码
- ❑ tidy.dir: 对某个目录下，所有 R 脚本进行格式化

1.4.2 formatR 安装

本节使用的系统环境是：

- ❑ Win7 64bit
- ❑ R: 3.0.1 x86_64-w64-mingw32/x64 b4bit

注 formatR 同时支持 Windows 7 环境和 Linux 环境。

formatR 的安装过程如下：

```
~ R  # 启动R程序
> install.packages("formatR")  # 安装formatR包
library(formatR)  # formatR加载
```

1.4.3 formatR 的使用

1. 字符串格式化

tidy.source() 函数，以字符串作为输入参数，对代码格式化。

```
> tidy.source(text = c("{if(TRUE)1 else 2; if(FALSE){1+1", "## comments", "} else 2}"))
{
    if (TRUE)
        1 else 2
    if (FALSE) {
        1 + 1
        ## comments
    } else 2
}
```

通过执行 tidy.source() 函数，把代码进行了重新格式化，让我们一眼就可以看得懂。

2. 文件格式化

messy.R 是一个不太规范的 R 程序文件。我们读入这个文件，然后通过 tidy.source() 函数，以文件对象作为输入参数，进行代码格式化。

```
> messy = system.file("format", "messy.R", package = "formatR")
> messy
[1] "C:/Program Files/R/R-3.0.1/library/formatR/format/messy.R"
```

messy.R 的原始代码输出：

```
> src = readLines(messy)
> cat(src,sep="\n")
    # a single line of comments is preserved
1+1

if(TRUE){
x=1  # inline comments
}else{
x=2;print('Oh no... ask the right bracket to go away!')}
1*3  # one space before this comment will become two!
2+2+2    # 'short comments'

lm(y~x1+x2, data=data.frame(y=rnorm(100),x1=rnorm(100),x2=rnorm(100)))
### only'single quotes' are allowed in comments
1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1  ## comments after a long line
'a character string with \t in it'

## here is a long long long long long long long long long long long long long long
 long long long long long long long comment
```

格式化后的代码输出：

```
> tidy.source(messy)
# a single line of comments is preserved
1 + 1
if (TRUE) {
    x = 1  # inline comments
} else {
    x = 2
    print("Oh no... ask the right bracket to go away!")
}
1 * 3  # one space before this comment will become two!
2 + 2 + 2  # 'short comments'

lm(y ~ x1 + x2, data = data.frame(y = rnorm(100), x1 = rnorm(100), x2 =
 rnorm(100)))  ### only 'single quotes' are allowed in comments
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1  ## comments after a long line
"a character string with \t in it"
## here is a long long long long long long long long long long long long long long
```

```
    long long long
    ##long long long long comment
```

可以看出，格式化后的输出，经过了空格、缩进、换行、注释等处理，代码可读性就增强了。

3. 格式化并输出 R 脚本文件

新建 R 脚本文件 demo.r。

```
~ vi demo.r
a<-1+1;a;matrix(rnorm(10),5);
if(a>2) { b=c('11',832);"#a>2";} else print('a is invalid!!')
```

格式化 demo.r。

```
> x = "demo.r"
> tidy.source(x)
a <- 1 + 1
a
matrix(rnorm(10), 5)
if (a > 2) {
    b = c("11", 832)
    "#a>2"
} else print("a is invalid!!")
```

输出格式化结果到文件 demo2.r，如图 1-3 所示。

```
> f="demo2.r"
> tidy.source(x, keep.blank.line = TRUE, file = f)
> file.show(f)
```

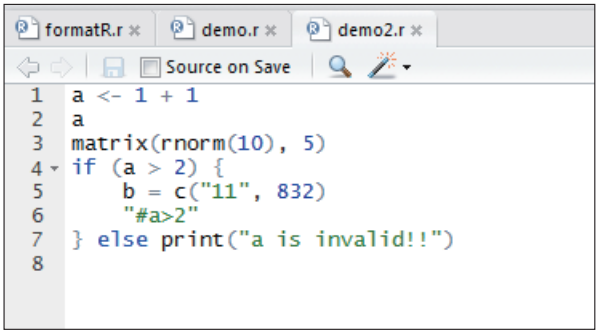


图 1-3 输出格式化结果到文件

4. 输出格式化代码和运行结果

使用 tidy.eval() 函数，以字符串形式，执行 R 脚本：

```
> tidy.eval(text = c("a<-1+1;a", "matrix(rnorm(10),5)"))
a <- 1 + 1
a
## [1] 2
matrix(rnorm(10), 5)
## [,1] [,2]
## [1,] 0.65050729 0.1725221
## [2,] 0.05174598 0.3434398
## [3,] -0.91056310 0.1138733
## [4,] 0.18131010 -0.7286614
## [5,] 0.40811952 1.8288346
```

这样直接在当前的运行环境中，就输出了代码和运行结果。

5. 格式化函数定义

通过 usage() 函数可以只打印出函数定义，跳过函数细节。以 var() 函数为例，输入 var，默认会打印出一个函数细节。

```
> var
function (x, y = NULL, na.rm = FALSE, use)
{
if (missing(use))
use <- if (na.rm)
"na.or.complete"
else "everything"
na.method <- pmatch(use, c("all.obs", "complete.obs", "pairwise.complete.obs",
"everything", "na.or.complete"))
if (is.na(na.method))
stop("invalid 'use' argument")
if (is.data.frame(x))
x <- as.matrix(x)
else stopifnot(is.atomic(x))
if (is.data.frame(y))
y <- as.matrix(y)
else stopifnot(is.atomic(y))
.Call(C_cov, x, y, na.method, FALSE)
}
<bytecode: 0x0000000008fad030>
<environment: namespace:stats>

> usage(var)  # 通过 usage，只打印函数定义
var(x, y = NULL, na.rm = FALSE, use)
```

有时候函数定义也很长，比如 lm() 函数，通过 usage 的 width 参数可以控制函数的显示宽度。

```
> usage(lm)
lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x =
FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)

> usage(lm,width=30)  # usage 的 width 参数，控制函数的显示宽度
lm(formula, data, subset, weights,
    na.action, method = "qr", model = TRUE,
    x = FALSE, y = FALSE, qr = TRUE,
    singular.ok = TRUE, contrasts = NULL,
    offset, ...)
```

6. GUI 工具

tidy.gui() 函数是一个 GUI 的工具，可以在界面上编辑并格式化 R 代码。首先安装 gWidgetsRGtk2 库：

```
> install.packages("gWidgetsRGtk2")
also installing the dependencies 'RGtk2', 'gWidgets'
```

打开 GUI 控制台：

```
> library("gWidgetsRGtk2")
> g = tidy.gui()
```

我们输入一段不太好看的代码，如图 1-4 所示。

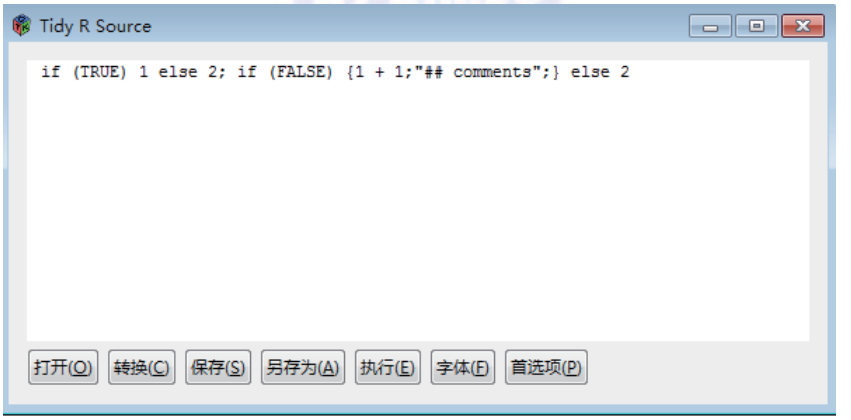


图 1-4 tidy 的 GUI

点击“转换”，结果如图 1-5 所示，可以看到，在 GUI 的编辑器中，R 语言的代码被格式化了！

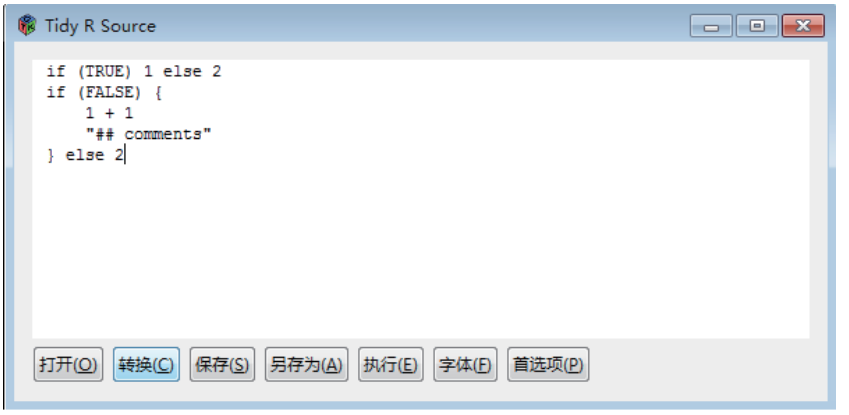


图 1-5　格式化后的代码

7. 格式化目录中的文件

tidy.dir() 函数可以批量格式化文件，对目录中的所有文件进行格式化操作。下面新建目录 :dir，并在目录 dir 中新建两个 R 脚本文件：dir.r, dir2.r。

```
~ mkdir dir  # 新建目录 dir
~ cd dir
~ vi dir.r  # 用 vi 新建文件 dir.r
a<-1+1;a;matrix(rnorm(10),5);

~ vi dir2.r
if(a>2) { b=c('11',832);"#a>2";} else print('a is invalid!!')
```

执行 tidy.dir：

```
> tidy.dir(path="dir")
tidying dir/dir.r
tidying dir/dir2.r
```

分别查看 dir.r 和 dir2.r：

```
~ vi dir.r
a <- 1 + 1
a
matrix(rnorm(10), 5)

~ vi dir2.r
if (a > 2) {
    b = c("11", 832)
    "#a>2"
} else print("a is invalid!!")
```

我们发现不规则的代码，已经被格式化了！

1.4.4 formatR 的源代码解析

通过上面的使用，我们不难发现，formatR 包的核心函数就是 tidy.source() 函数，从 Github 上面找到源代码：https://github.com/yihui/formatR/blob/master/R/tidy.R。我将在代码中增加注释：

```
tidy.source = function(
  source = 'clipboard', keep.comment = getOption('keep.comment', TRUE),
  keep.blank.line = getOption('keep.blank.line', TRUE),
  replace.assign = getOption('replace.assign', FALSE),
  left.brace.newline = getOption('left.brace.newline', FALSE),
  reindent.spaces = getOption('reindent.spaces', 4),
  output = TRUE, text = NULL,
  width.cutoff = getOption('width'), ...
) {

   if (is.null(text)) {   # 判断输入来源为剪贴板
    if (source == 'clipboard' && Sys.info()['sysname'] == 'Darwin') {
      source = pipe('pbpaste')
    }
  } else {  # 判断输入来源为字符串
    source = textConnection(text); on.exit(close(source))
  }

 text = readLines(source, warn = FALSE)  # 按行读取来源数据
  if (length(text) == 0L || all(grepl('^\\s*$', text))) {   # 文件行数判断
    if (output) cat('\n', ...)
    return(list(text.tidy = text, text.mask = text))
  }

    if (keep.blank.line && R3) {  # 空行处理
    one = paste(text, collapse = '\n') # record how many line breaks before/after
    n1 = attr(regexpr('^\n*', one), 'match.length')
    n2 = attr(regexpr('\n*$', one), 'match.length')
  }

  if (keep.comment) text = mask_comments(text, width.cutoff, keep.blank.line)
  # 注释处理
  text.mask = tidy_block(text, width.cutoff, replace.assign && length(grep('=', text)))
  # 把输入的 R 代码，先转成表达式，再转回字符串。用来实现对每个语句的截取
  text.tidy = if (keep.comment) unmask.source(text.mask) else text.mask
  # 对注释排版
  text.tidy = reindent_lines(text.tidy, reindent.spaces)  # 重新定位缩进
```

```
  if (left.brace.newline) text.tidy = move_leftbrace(text.tidy)    # 扩号换行
  if (keep.blank.line && R3) text.tidy = c(rep('', n1), text.tidy, rep('', n2))
  # 增加首尾空行
  if (output) cat(paste(text.tidy, collapse = '\n'), '\n', ...)
  # 在 console 打印格式化后的结果
  invisible(list(text.tidy = text.tidy, text.mask = text.mask))
  # 返回，但不打印结果
}
```

1.4.5　源代码中的 Bug

在读源代码的过程中，我发现有一个小问题，即在 R 3.0.1 版本，没有对向右赋值操作 (->) 进行处理。我已经就这个问题给作者提 Bug 了，参见 https://github.com/yihui/formatR/issues/31。Bug 测试代码如下：

```
> c('11',832)->x2
> x2
[1] "11"  "832"

> tidy.source(text="c('11',832)->x2")   # 格式化代码
c("11", 832) <- x2

> tidy.eval(text="c('11',832)->x2")
c("11", 832) <- x2
Error in eval(expr, envir, enclos) : object 'x2' not found
```

Bug 已修复。作者回复：“这个问题已经在 R 3.0.2 中修正了。”

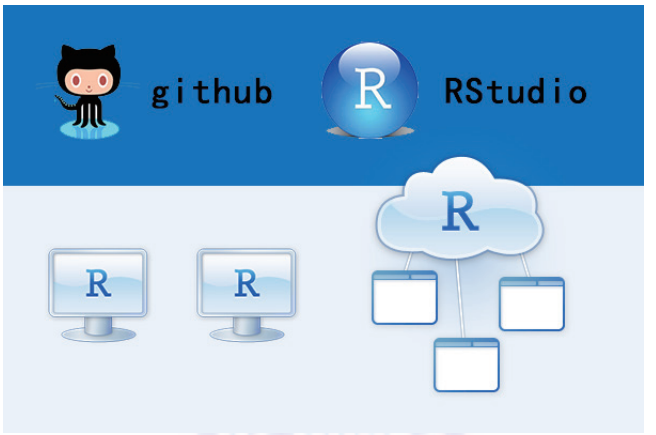
```
> formatR::tidy.source(text="c('11',832)->x2") # 格式化代码
x2 <- c("11", 832)

> sessionInfo()
R version 3.0.2 (2013-09-25)
Platform: x86_64-pc-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
[1] formatR_0.10.3
```

formatR 包提供的功能非常实用，特别是读别人写的不规范的代码的时候。建议各 IDE 厂商能把 formatR 作为标准的格式化工具直接嵌入编辑器的工具中。让我们把阅读别人的代码，也变成一件快乐的事情吧。

1.5 多人在线协作 R 开发 RStudio Server

问题

R 语言开发，哪个工具最好用？



引言

RStudio 是 R 语言开发中的利器，是最好用的 R 语言 IDE 集成环境。RStudio Server 更是利器中的神器。不仅提供了 Web 的功能，可以安装到远程服务器上，通过 Web 进行访问，还支持多用户的协作开发。如此神器，快来动手试一下吧。

1.5.1 RStudio 和 RStudio Server

RStudio 是一个强大的、免费的、开源的 R 语言集成开发环境的应用软件，可以安装在 Windows、Linux 和 Mac 不同操作系统上。RStudio Server 是一个基于 Web 访问的 RStudio 云端开发环境，需要安装在 Linux 服务器上面，支持多用户远程访问使用。

1.5.2 RStudio Server 安装

本文使用的系统环境是：

- Linux: Ubuntu Server 12.04.2 LTS 64bit

❑ R: 3.0.1 x86_64-pc-linux-gnu

❑ IP: 192.168.1.13

注　RStudio Server 只支持 Linux 系统环境。下载最新版本 RStudio Server 的地址是 http://www.rstudio.com/ide/download/server.html。

在 Linux Ubuntu 环境中，下载并安装 64 位的 Rstudio Server：

```
~ sudo apt-get install gdebi-core
~ sudo apt-get install libapparmor1  # Required only for Ubuntu, not Debian
~ wget http://download2.rstudio.org/rstudio-server-0.97.551-amd64.deb
~ sudo gdebi rstudio-server-0.97.551-amd64.deb
```

安装后，RStudio Server 会自动启动运行。

```
~ ps -aux|grep rstudio-server  # 查看 RStudio Server 运行进程
998       2914  0.0  0.1 192884  2568 ?        Ssl  10:40   0:00 /usr/lib/
rstudio-server/bin/rserver
```

可以看到，RStudio Server 的服务已启动，8787 端口被打开。

1.5.3　RStudio Server 使用

通过浏览器，我们访问 RStudio Server: http://192.168.1.13:8787，IP 地址为 RStudio Server 服务器的地址，如图 1-6 所示。

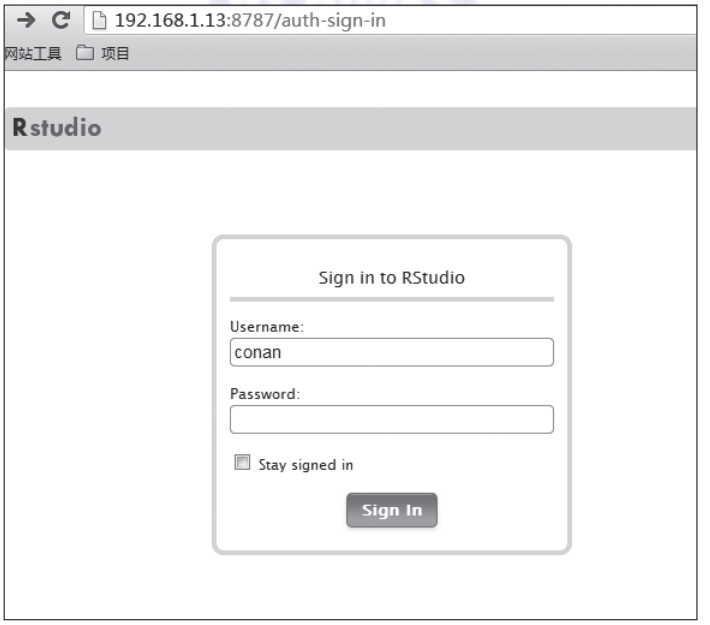


图 1-6　RStudio Server 登录界面

RStudio Server 登录需要用 Linux 系统的用户账号。如果想增加或减少用户，直接对 Linux 系统用户进行操作就可以了。我的环境中用户登录，用户名是 conan，密码是 conan111。登录之后看到的界面如图 1-7 所示。

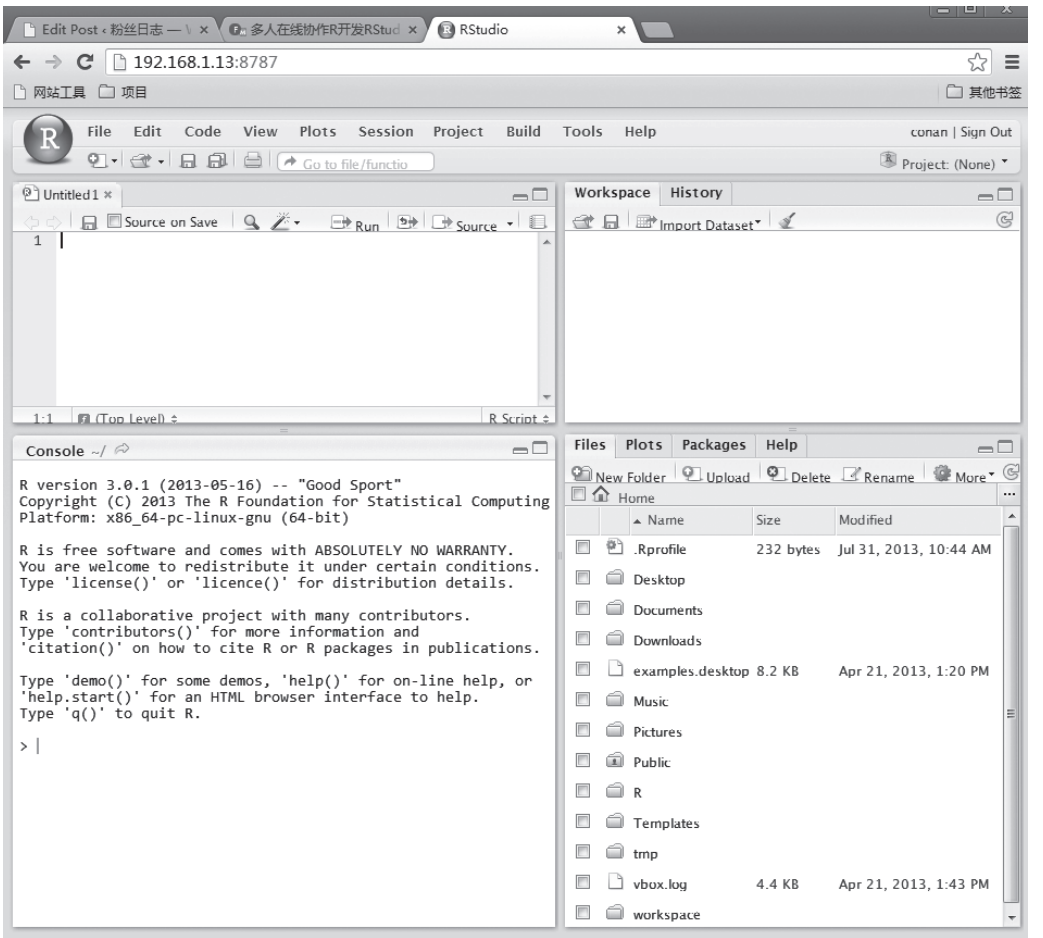


图 1-7 RStudio Server 的 Web 界面

1. RStudio Server 的系统配置

RStudio Server 主要有两个配置文件，默认文件不存在。

```
/etc/rstudio/rserver.conf
/etc/rstudio/rsession.conf
```

设置端口和 ip 控制：

```
~ vi /etc/rstudio/rserver.conf
```

```
www-port=8080  # 监听端口
www-address=127.0.0.1  # 允许访问的 IP 地址，默认为 0.0.0.0
```

重启 Rstudio Server 服务器，配置生效：

```
~ sudo rstudio-server restart
```

会话配置管理：

```
~ vi /etc/rstudio/rsession.conf
session-timeout-minutes=30  # 会话超时时间
r-cran-repos=http://ftp.ctex.org/mirrors/CRAN/  # 设置 CRAN 资源库
```

2. RStudio Server 的系统管理

启动、停止、重启 RStudio Server 服务器的命令如下：

```
~ sudo rstudio-server start    # 启动
~ sudo rstudio-server stop     # 停止
~ sudo rstudio-server restart  # 重启
```

查看运行中的 R 进程：

```
~ sudo rstudio-server active-sessions
PID     TIME COMMAND
6817 00:00:03 /usr/lib/rstudio-server/bin/rsession -u zd
```

指定 PID, 停止运行中的 R 进程：

```
~ sudo  rstudio-server suspend-session 6817
~ sudo  rstudio-server active-sessions  # 再次查看进程
  PID     TIME COMMAND
```

停止所有运行中的 R 进程：

```
~ sudo rstudio-server suspend-all
```

强制停止运行中的 R 进程，此操作优先级最高，立刻执行。

```
~ sudo rstudio-server force-suspend-session <pid>
~ sudo rstudio-server force-suspend-all
```

RStudio Server 临时下线，不允许 Web 访问，并给用户友好的错误提示：

```
~ sudo rstudio-server offline
rstudio-server start/running, process 6880
```

RStudio Server 上线：

```
~ sudo rstudio-server online
rstudio-server start/running, process 6908
```

RStudio Server 的其他操作和单机版的 RStudio 一样。

1.5.4 RStudio Server 多人协作

1. 增加新用户和新用户组

```
~ sudo groupadd hadoop  # 创建 Hadoop 用户组
~ sudo useradd hadoop -g hadoop  # 创建 Hadoop 用户并加入到 Hadoop 用户组
~ sudo passwd hadoop  # 设置 Hadoop 用户的密码
~ sudo adduser hadoop sudo  # 增加 Hadoop 用户到 sudo 组
~ sudo mkdir /home/hadoop  # 创建 Hadoop 用户的 home 目录
~ sudo chown -R hadoop:hadoop /home/hadoop  # 给 /home/hadoop 目录及子目录，设置用户权限
```

以 Hadoop 用户登录，检查用户是否设置成功

```
~ ssh hadoop@localhost  # 通过 ssh 远程登录
~ bash
~ pwd  # 查看登录后的访问目录
/home/hadoop
```

新打开浏览器窗口通过 Hadoop 账号登录，如图 1-8 所示。

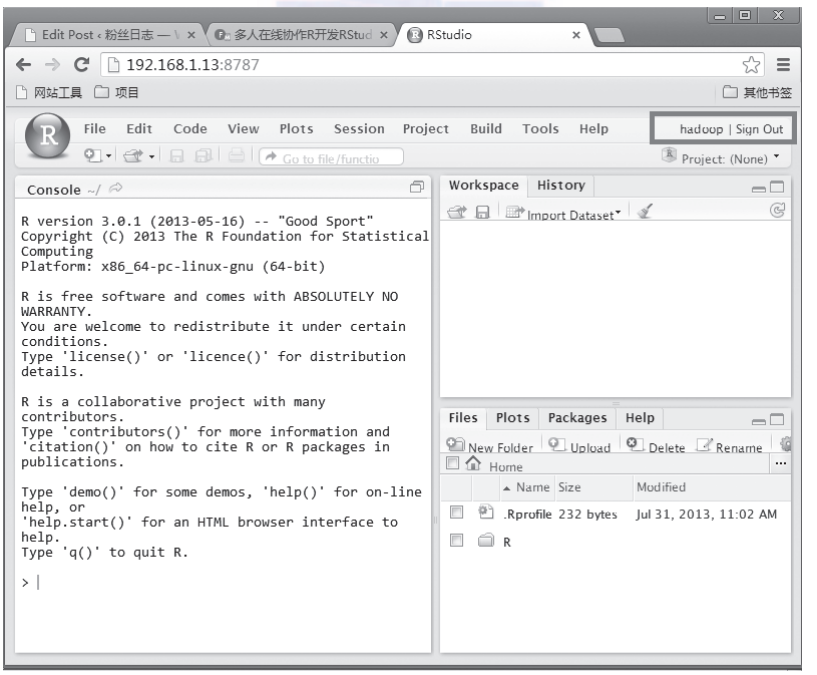


图 1-8 以 Hadoop 账号登录

2. Git 代码共享

首先安装 Git：

```
~ sudo apt-get install git
~ ssh-keygen -t rsa  # 生成 rsa 密钥对
~ cat /home/conan/.ssh/id_rsa.pub  # 查看公钥
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDMmnFyZe2RHpXaGmENdH9kSyDyVzRas4GtRwMNx+qQ
4QsB8xVTrIbFayG2ilt+P8UUkVYO0qtUJIaLRjGy/SvQzzL7JKX12+VyYoKTfKvZZnANJ414d6oZpbDw
sC0Z7JARcWsFyTW1KxOMyesmzNNdB+F3bYN9sYNiTkOeVNVYmEQ8aXywn4kcljBhVpT8PbuHl5eadSLt
5zpN6bcX7tlquuTlRpLi1e4K+8jQo67H54FuDyrPLUYtVaiTNT/xWN6IU+DQ9CbfykJ0hrfDU1d1LiLQ
4K2Fdg+vcKtB7Wxez2wKjsxb4Cb8TLSbXdIKEwSOFooINw25g/Aamv/nVvW1 conan@conan-deskop
```

接下来，我们需要把本地项目上传到 Github。首先在 Github 上创建一个新的项目 rstudio-demo，地址为 https://github.com/bsspirit/rstudio-demo，通过下面的操作上传本地目录到 rstudio-demo 项目。

```
~ mkdir /home/conan/R/github    # 创建 rstudio-demo 项目目录
~ cd /home/conan/R/github
~ git init  # 初始化 Git
~ git add   # 增加当前目录及子目录到本地 Git 库
~ git commit -m 'first comment'  # 在本地 Git 库提交
~ git remote add origin git@github.com:bsspirit/rstudio-demo.git
# 绑定当前目录和 github 的项目
~ git push -u origin master  # 上传本地 Git 库中的代码到 Github
```

打开 RStudio 设置到 /home/conan/R/github 目录，tools–>version control –> project setup，如图 1-9 所示。

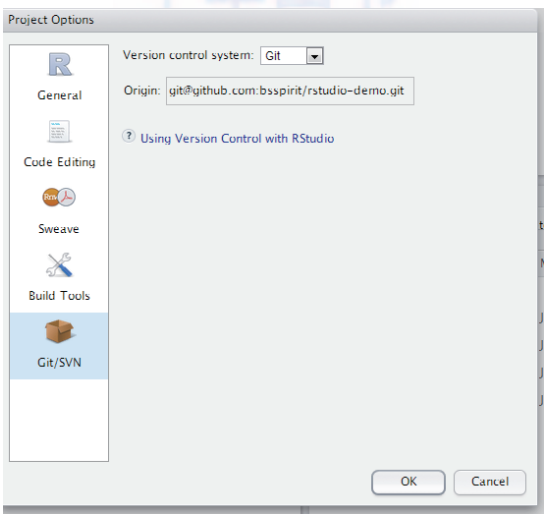


图 1-9　在 RStudio 中，配置 Github 地址

在 RStudio 中修改 sayHello.r 的代码：

```
sayHello<-function(name){
  print(paste("hello",name))
}

sayHello("Conan")
sayHello("World")
```

点击 tools–>version control–> commit 提交，如图 1-10 所示。

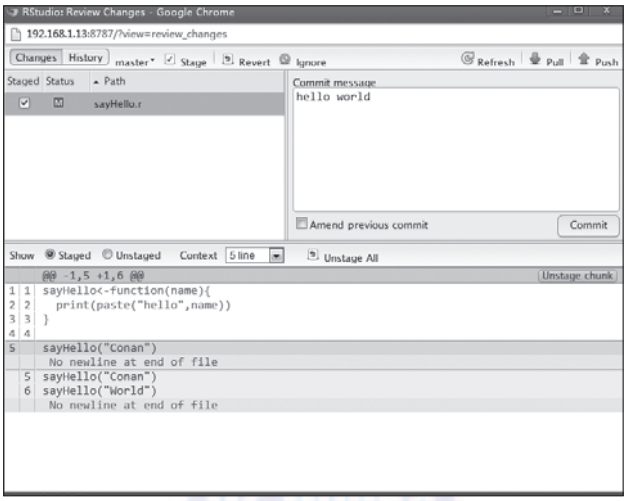


图 1-10 通过 RStudio 进行 Git 操作

上传到 Github，只需要点击 tools–>version control–> push，如图 1-11 所示。

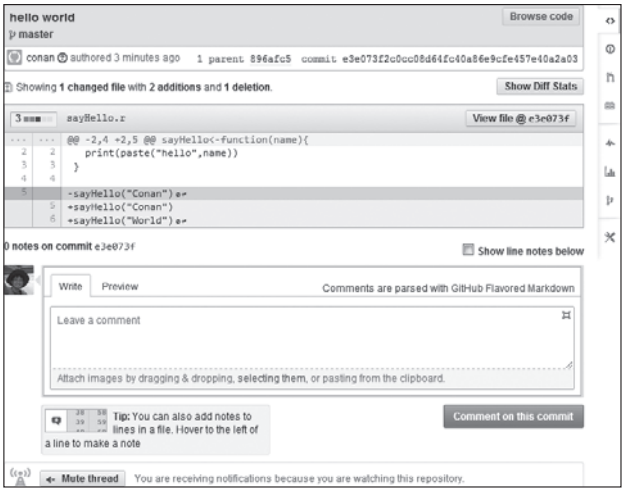


图 1-11 在 Github 上查看提交的操作

RStudio 有如此强大的功能，极大地降低了编程的门槛。还没有用过的同学，赶紧去体验一把极客的感觉吧！

1.6　R 和 JSON 的傻瓜式编程

问题

如何让 R 语言的数据类型转换成 JSON 数据类型？

引言

JSON 作为一种轻量级数据格式，被大量地应用在各种程序环境中。JSON(JavaScript Object Notation) 是 JavaScript 的内嵌的标准对象，同时也是 MongoDB 的表结构存储类型。JSON 是半结构化的，可以表达出丰富的文档含义。JSON 文档比 XML 文档要少很多，更适合于网络传输。早期 R 语言编程很少会用到 JSON，但随着 R 语言的壮大，R 也在伸向各种领域，JSON 就是与其他领域的一个交点。如何让 R 语言傻瓜式转型 JSON 呢？请看下文介绍。

1.6.1　rjson 包介绍

rjson 是一个 R 语言与 JSON 进行转换的包，非常简单，支持 R 自身语言转型和基于 C 类库的转型两种方式。rjson 包提供的函数只有 3 个，fromJSON(), newJSONParser(), toJSON()。后面我们将介绍如何使用 rjson 包。本节使用的系统环境是：

- Windows 7: x86_64-w64-mingw32/x64 (64-bit)
- R: version 3.0.1

注　rjson 同时支持 Windows 7 环境和 Linux 环境。

1. 安装并加载 rjson

```
> install.packages("rjson")  # 安装 rjson
> library(rjson)  # 加载 rjson
```

接下来，我们进行测试。新建 JSON 文件 fin0.json：

```
~ vi fin0.json

{
    "table1": {
        "time": "130911",
        "data": {
            "code": [
                "TF1312",
                "TF1403",
                "TF1406"
            ],
            "rt_time": [
                130911,
                130911,
                130911
            ]
        }
    },
    "table2": {
        "time": "130911",
        "data": {
            "contract": [
                "TF1312",
                "TF1312",
                "TF1403"
            ],
            "jtid": [
                99,
                65,
                21
            ]
        }
    }
}
```

2. 调用函数 fromJSON(): 从 JSON 到 R

从 fin0.json 文件中，读取 JSON 并解析成 R 语言对象。我们通常把字节或者文字格式转型为程序对象的过程叫反序列化过程，与之相反的过程，叫做序列化过程。

```
> json_data <- fromJSON(paste(readLines("fin0.json"), collapse=""))
> json_data
$table1
$table1$time
[1] "130911"

$table1$data
$table1$data$code
[1] "TF1312" "TF1403" "TF1406"

$table1$data$rt_time
[1] 130911 130911 130911

$table2
$table2$time
[1] "130911"

$table2$data
$table2$data$contract
[1] "TF1312" "TF1312" "TF1403"

$table2$data$jtid
[1] 99 65 21
```

检查转型后，对应 R 语言的数据类型：

```
> class(json_data)
[1] "list"
> class(json_data$table2)
[1] "list"
> class(json_data$table2$data)
[1] "list"
> class(json_data$table2$data$jtid)
[1] "numeric"
> class(json_data$table1$data$code)
[1] "character"
```

我们看到原 JSON 对象转型后，除最内层外，其他都被解析为 R 的列表 (list) 类型，最内层则是基本 (numeric,character) 类型。在 R 对象结构中取 JSON 数据的一个叶子节点，JSON 的索引路径为 json.table1.data.code[0]。

```
> json_data$table1$data$code
[1] "TF1312" "TF1403" "TF1406"
> json_data$table1$data$code[1]
[1] "TF1312"
```

3. toJSON() 从 R 到 JSON

把 R 对象转型为 JSON 串，这个过程叫做序列化过程。还以刚才的 json_data 为例。

```
> json_str<-toJSON(json_data)

> print(json_str)
[1] "{\"table1\":{\"time\":\"130911\",\"data\":{\"code\":[\"TF1312\",\"TF1403\",
\"TF1406\"],\"rt_time\":[130911,130911,130911]}},\"table2\":{\"time\":\"130911\"
,\"data\":{\"contract\":[\"TF1312\",\"TF1312\",\"TF1403\"],\"jt
id\":[99,65,21]}}}"

> cat(json_str)
{"table1":{"time":"130911","data":{"code":["TF1312","TF1403","TF1406"],"rt_time"
:[130911,130911,130911]}},"table2":{"time":"130911","data":{"contract":["TF1312",
"TF1312","TF1403"],"jtid":[99,65,21]}}}
```

我们只要使用 toJSON() 函数，就可以实现 R 对象向 JSON 的转型。如果用 print() 函数输出，结果就是带转义的输出 (\")；如果直接用 cat 函数输出，结果就是标准的 JSON 串格式。把 JSON 输出到文件 fin0_out.json, 有 2 种方法，即 writeLines() 和 sink()。

```
> writeLines(json_str, "fin0_out1.json")  # writeLines 方法
> sink("fin0_out2.json")  # sink 方法
> cat(json_str)
> sink()
```

虽然写法不同，但是输出结果是一个样的，writeLines 最后新建一个空行。

```
{"table1":{"time":"130911","data":{"code":["TF1312","TF1403","TF1406"],"rt_time":
[130911,130911,130911]}},"table2":{"time":"130911","data":{"contract":["TF1312",
"TF1312","TF1403"],"jtid":[99,65,21]}}}
```

4. C 语言库和 R 语言库转型，并进行性能测试

我们对 fromJSON 进行性能测试：

```
> system.time( y <- fromJSON(json_str,method="C") )
用户 系统 流逝
  0   0    0
> system.time( y2 <- fromJSON(json_str,method = "R") )
 用户   系统   流逝
0.02  0.00 0.02
> system.time( y3 <- fromJSON(json_str) )
用户 系统 流逝
  0   0    0
```

我们可以看到，基于 C 语言库的操作比基于 R 语言库的要快，因为数据量很小，所以 0.02 并不明显。当 JSON 串很大的时候，这个差距就会变得相当大了。fromJSON 默认使用 C 语言库的方法，所以我们平时处理不用加 method='C' 的参数。下面做 toJSON 的性能测试。

```
> system.time( y <- toJSON(json_data,method="C") )
用户 系统 流逝
  0    0   0
> system.time( y2 <- toJSON(json_data,method = "R") )
 用户   系统 流逝
0.02 0.00 0.01
> system.time( y3 <- toJSON(json_data) )
用户 系统 流逝
  0    0   0
```

解释同前。

1.6.2　RJSONIO 包介绍

RJSONIO 包，提供了 2 个主要的操作，把 JSON 串反序列化成 R 对象，把 R 对象序列化成 JSON 串。RJSONIO 包的两个主要函数是 fromJSON(), toJSON()，它还包括几个辅助函数，即 asJSVars(), basicJSONHandler(), Bob(), isValidJSON(), readJSONStream()。RJSONIO 包解决了 rjson 包序列化大对象慢的问题。RJSONIO 依赖于底层的 C 语言类库 libjson。

1. 安装并加载 RJSONIO

```
> install.packages("RJSONIO")
> library(RJSONIO)
```

2. fromJSON() 从 JSON 到 R

同 rjson 一样，测试 fromJSON() 函数。

```
> json_data <- fromJSON(paste(readLines("fin0.json"), collapse=""))
> json_data
$table1
$table1$time
[1] "130911"

$table1$data
$table1$data$code
[1] "TF1312" "TF1403" "TF1406"

$table1$data$rt_time
[1] 130911 130911 130911
```

```
$table2
$table2$time
[1] "130911"
$table2$data
$table2$data$contract
[1] "TF1312" "TF1312" "TF1403"
$table2$data$jtid
[1] 99 65 21
```

我们发现与 rjson 的结果是一样，R 对象除最内层外，其他都是列表 (list) 类型。下面取叶子节点：

```
> json_data$table1$data$code
[1] "TF1312" "TF1403" "TF1406"
> json_data$table1$data$code[1]
[1] "TF1312"
```

3. toJSON() 从 R 到 JSON

做 toJSON 的性能测试：

```
> json_str<-toJSON(json_data)
> print(json_str)
[1] "{\n \"table1\": {\n \"time\": \"130911\",\n\"data\": {\n \"code\":
[\"TF1312\", \"TF1403\", \"TF1406\" ],\n\"rt_time\": [ 1.3091e+05, 1.3091e+05,
1.3091e+05 ] \n} \n},\n\"table2\": {\n \"time\": \"130911\",\n\"data\": {\n
\"contract\": [ \"TF1312\", \"TF1312\", \"TF1403\" ],\n\"jtid\": [99,
65,21 ] \n} \n} \n}"

> cat(json_str)
{
 "table1": {
 "time": "130911",
"data": {
 "code": [ "TF1312", "TF1403", "TF1406" ],
"rt_time": [ 1.3091e+05, 1.3091e+05, 1.3091e+05 ]
}
},
"table2": {
 "time": "130911",
"data": {
 "contract": [ "TF1312", "TF1312", "TF1403" ],
"jtid": [99,65,21 ]
}
}
}
```

toJSON 的函数输出与 rjson 是不一样的，这个输出是格式化的。下面输出到文件：

```
> writeLines(json_str, "fin0_io.json")
```

文件结果：

```
{
 "table1": {
 "time": "130911",
"data": {
 "code": [ "TF1312", "TF1403", "TF1406" ],
"rt_time": [ 1.3091e+05, 1.3091e+05, 1.3091e+05 ]
}
},
"table2": {
 "time": "130911",
"data": {
 "contract": [ "TF1312", "TF1312", "TF1403" ],
"jtid": [     99,     65,     21 ]
}
}
}
```

4. isValidJSON() 验证 JSON 是否合法

验证 JSON 的格式是否合法。

```
> isValidJSON(json_str)
Error in file(con, "r") : cannot open the connection

> isValidJSON(json_str,TRUE)  # 合法 JSON
[1] TRUE

> isValidJSON(I('{"foo" : "bar"}'))  # 合法 JSON
[1] TRUE
> isValidJSON(I('{foo : "bar"}'))  # 不合法 JSON
[1] FALSE
```

5. asJSVars() 转换为 JavaScript 变量格式

```
> cat(asJSVars( a = 1:10, myMatrix = matrix(1:15, 3, 5)))
a = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ] ;

myMatrix = [ [ 1, 4, 7, 10, 13 ],
           [ 2, 5, 8, 11, 14 ],
           [ 3, 6, 9, 12, 15 ] ] ;
```

得到两个 JavaScript 变量，即数组 a 和二维数组 myMatrix.

1.6.3 自定义 JSON 的实现

在 R 语言中，我们最常用的类型是 data.frame，接下来我们自己实现 data.frame 类型和 JSON 的转换。首先，把 R 的 data.frame 对象转成我们定义的 JSON 格式。

定义输出 JSON 的格式：

```
[
    {
        "code": "TF1312",
        "rt_time": "152929",
        "rt_latest": 93.76,
        "rt_bid1": 93.76,
        "rt_ask1": 90.76,
        "rt_bsize1": 2,
        "rt_asize1": 100,
        "optionValue": -0.4,
        "diffValue": 0.6
    }
]
```

定义 R 语言 data.frame 类型对象：

```
> df<-data.frame(
  code=c('TF1312','TF1310','TF1313'),
  rt_time=c("152929","152929","152929"),
  rt_latest=c(93.76,93.76,93.76),
  rt_bid1=c(93.76,93.76,93.76),
  rt_ask1=c(90.76,90.76,90.76),
  rt_bsize1=c(2,3,1),
  rt_asize1=c(100,1,11),
  optionValue=c(-0.4,0.2,-0.1),
  diffValue=c(0.6,0.6,0.5)
)

> df
    code rt_time rt_latest  rt_bid1  rt_ask1 rt_bsize1 rt_asize1 optionValue diffValue
1 TF1312  152929     93.76   93.76   90.76         2       100        -0.4       0.6
2 TF1310  152929     93.76   93.76   90.76         3         1         0.2       0.6
3 TF1313  152929     93.76   93.76   90.76         1        11        -0.1       0.5
```

直接使用 toJSON，输出的 JSON 串按列组合成了数组，并不是我们想要的。

```
> cat(toJSON(df))
{
```

```
 "code": ["TF1312", "TF1310", "TF1313"],
"rt_time": ["152929", "152929", "152929"],
"rt_latest": [93.76, 93.76, 93.76],
"rt_bid1": [93.76, 93.76, 93.76],
"rt_ask1": [90.76, 90.76, 90.76],
"rt_bsize1": [2, 3, 1],
"rt_asize1": [100, 1, 11],
"optionValue": [-0.4, 0.2, -0.1],
"diffValue": [0.6, 0.6, 0.5]
}
```

我们要对 data.frame 类型进行数据处理：

```
> library(plyr)  #用 plyr 进行数据转换
> cat(toJSON(unname(alply(df, 1, identity))))
[
 {
 "code": "TF1312",
"rt_time":  "152929",
"rt_latest":  93.76,
"rt_bid1":    93.76,
"rt_ask1":    90.76,
"rt_bsize1":      2,
"rt_asize1":    100,
"optionValue": -0.4,
"diffValue":    0.6
},
{
 "code": "TF1310",
"rt_time": "152929",
"rt_latest": 93.76,
"rt_bid1":   93.76,
"rt_ask1":   90.76,
"rt_bsize1":     3,
"rt_asize1":     1,
"optionValue": 0.2,
"diffValue":   0.6
},
{
 "code": "TF1313",
"rt_time": "152929",
"rt_latest": 93.76,
"rt_bid1":   93.76,
"rt_ask1":   90.76,
"rt_bsize1":     1,
"rt_asize1":    11,
"optionValue":-0.1,
```

```
"diffValue":   0.5
}
]
```

输出的结果已经通过 alply() 函数做了数据变换，这正是我希望的按行输出的结果。

1.6.4 JSON 性能比较

性能比较我们做 2 组测试，一组是 rjson 和 RJSONIO 对大对象进行序列化 (toJSON) 测试，另一组是 RJSONIO 包序列化 (toJSON) 列式输出和行式输出的测试。

1. rjson 和 RJSONIO 对大对象进行序列化 (toJSON) 测试

创建一个 rjson 测试脚本，在命令行运行。

```
> library(rjson)
> df<-data.frame(
+   a=rep(letters,10000),
+   b=rnorm(260000),
+   c=as.factor(Sys.Date()-rep(1:260000))
+ )
> system.time(rjson::toJSON(df))
1.01 0.02 1.03
> system.time(rjson::toJSON(df))
1.01 0.03 1.04
> system.time(rjson::toJSON(df))
0.98 0.05 1.03
```

同样，再创建一个 RJSONIO 测试脚本，在命令行运行。

```
> library(RJSONIO)
> df<-data.frame(
+   a=rep(letters,10000),
+   b=rnorm(260000),
+   c=as.factor(Sys.Date()-rep(1:260000))
+ )
> system.time(RJSONIO::toJSON(df))
2.23 0.02 2.24
> system.time(RJSONIO::toJSON(df))
2.30 0.00 2.29
> system.time(RJSONIO::toJSON(df))
2.25 0.01 2.26
```

对比结果发现，rjson 的性能优于 RJSONIO。

2. rjson 和 RJSONIO，序列化 (toJSON) 列式输出和行式输出的测试

创建一个 rjson 测试脚本，在命令行运行。

```
> library(rjson)
> library(plyr)
> df<-data.frame(
+   a=rep(letters,100),
+   b=rnorm(2600),
+   c=as.factor(Sys.Date()-rep(1:2600))
+ )
> system.time(rjson::toJSON(df))
0.01 0.00 0.02
> system.time(rjson::toJSON(df))
0.01 0.00 0.02
> system.time(rjson::toJSON(unname(alply(df, 1, identity))))
1.55 0.02 1.56
> system.time(rjson::toJSON(unname(alply(df, 1, identity))))
0.83 0.00 0.83
```

同样，再创建一个 RJSONIO 测试脚本，在命令行运行。

```
> library(RJSONIO)
> library(plyr)
> df<-data.frame(
+   a=rep(letters,100),
+   b=rnorm(2600),
+   c=as.factor(Sys.Date()-rep(1:2600))
+ )
> system.time(RJSONIO::toJSON(df))
0.03 0.00 0.03
> system.time(RJSONIO::toJSON(df))
0.04 0.00 0.03
> system.time(RJSONIO::toJSON(unname(alply(df, 1, identity))))
2.82 0.02 2.84
> system.time(RJSONIO::toJSON(unname(alply(df, 1, identity))))
2.06 0.00 2.06
```

通过测试我们发现，用 toJSON 直接列式输出，比行式输出要高效得多，这是因为行输出需要多一步数据处理的过程。这里说 rjson 比 RJSONIO 高效，而前面说 RJSONIO 包解决了 rjson 包序列化大对象慢的问题，似乎有些矛盾。当然，我的一次的测试不能证明这个结论的，也希望大家在有条件、有精力的情况下，自己做一些测试。

1.7 R语言的高质量图形渲染库Cairo

问题

如何让R语言画出无锯齿的高清图?

引言

R语言不仅在统计分析和数据挖掘领域计算能力强大，它在数据可视化领域也不逊于昂贵的商业软件。当然，R在可视化上强大，其背后离不开各种开源软件包的支持，Cairo就是这样一个用于矢量图形处理的类库。Cairo可以创建高质量的矢量图形(GIF、SVG、PDF、PostScript)和位图(PNG、JPEG、TIFF)，同时支持在后台程序中高质量渲染！本节将介绍Cairo在R语言中的使用。

1.7.1 Cairo介绍

Cairo是一个用于图形绘图和渲染的免费库，支持复杂的2D的绘图功能，支持硬件加速。虽然，Cairo是用C语言编写的，但提供多种语言的接口，允许其他语言直接调用，包括有C++、C#、Java、Python、Perl、Ruby、Scheme、Smalltalk等语言。Cairo发布的许可协议为GNU Lesser General Public License version 2.1(LGPL)或Mozilla Public License 1.1(MPL)。R语言Cairo接口的官方发布页是http://www.rforge.net/Cairo/。

1.7.2 Cairo包安装

本节使用的系统环境是：

- Linux: Ubuntu 12.04.2 LTS 64bit
- R: 3.0.1 x86_64-pc-linux-gnu

注　caTools 同时支持 Windows 7 环境和 Linux 环境。

Cairo 包在 Linux Ubuntu 系统中的安装过程如下：

```
~ sudo apt-get install libcairo2-dev  # Cairo 的底层依赖库
~ sudo apt-get install libxt-dev
~ R  # 启动 R 程序
> install.packages("Cairo")  # 安装 Cairo 包
```

1.7.3　Cairo 使用

Cairo 使用起来非常简单，和基础包 grDevices 中的函数对应。

- ❑ CairoPNG: 对应 grDevices:png()。
- ❑ CairoJPEG: 对应 grDevices:jpeg()。
- ❑ CairoTIFF: 对应 grDevices:tiff()。
- ❑ CairoSVG: 对应 grDevices:svg()。
- ❑ CairoPDF: 对应 grDevices:pdf()。

我常用的图形输出，就是 png 和 svg。下面检查 Cairo 的兼容性：

```
> library(Cairo)  # 加载 Cairo 包
> Cairo.capabilities()  # 检查 Cairo 包支持的图片格式
   png   jpeg   tiff    pdf    svg     ps    x11    win raster
  TRUE   TRUE  FALSE   TRUE   TRUE   TRUE   TRUE  FALSE   TRUE
```

从兼容性的检查结果，我们可以查看 Cairo 支持的图形输出格式：

- ❑ 支持：png、jpeg、pdf、svg、ps、x11(Linux 桌面)、raster
- ❑ 不支持：tiff、win(win 桌面)

注　如果是 Windows 系统，则 x11 为 FALSE, win 为 TRUE。

下面比较 CairoPNG() 和 png() 的输出效果。

1. 散点图

首先我们来画一个 6000 个点的散点图。

```
> x<-rnorm(6000)  # 随机取 6000 个点坐标
> y<-rnorm(6000)

> png(file="plot4.png",width=640,height=480)  # png 函数
> plot(x,y,col="#ff000018",pch=19,cex=2,main = "plot")
> dev.off()

> CairoPNG(file="Cairo4.png",width=640,height=480)  # CairoPNG 函数
> plot(x,y,col="#ff000018",pch=19,cex=2,main = "Cairo")
> dev.off()
```

在当前目录，会生成 2 个 png 文件，即 plot4.png 和 Cairo4.png，见图 1-12 和图 1-13。

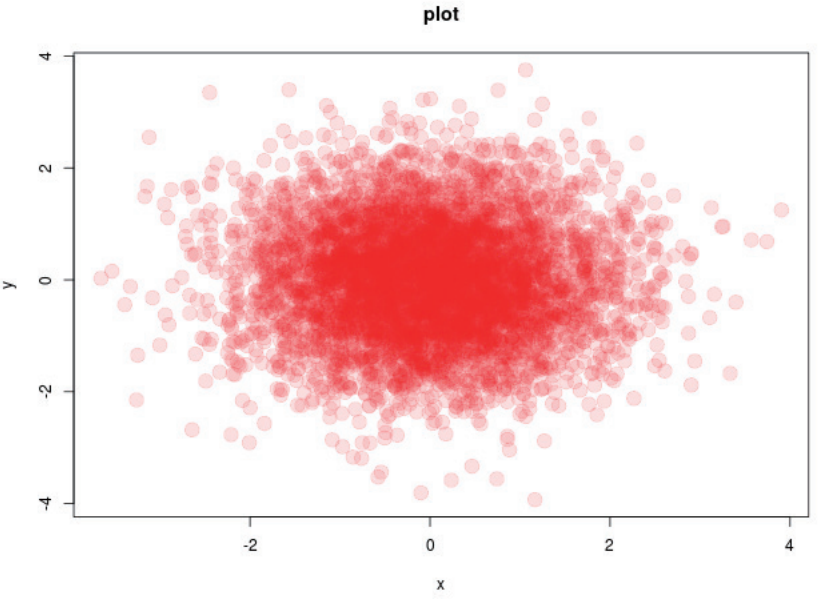


图 1-12　以 png() 函数生成的图

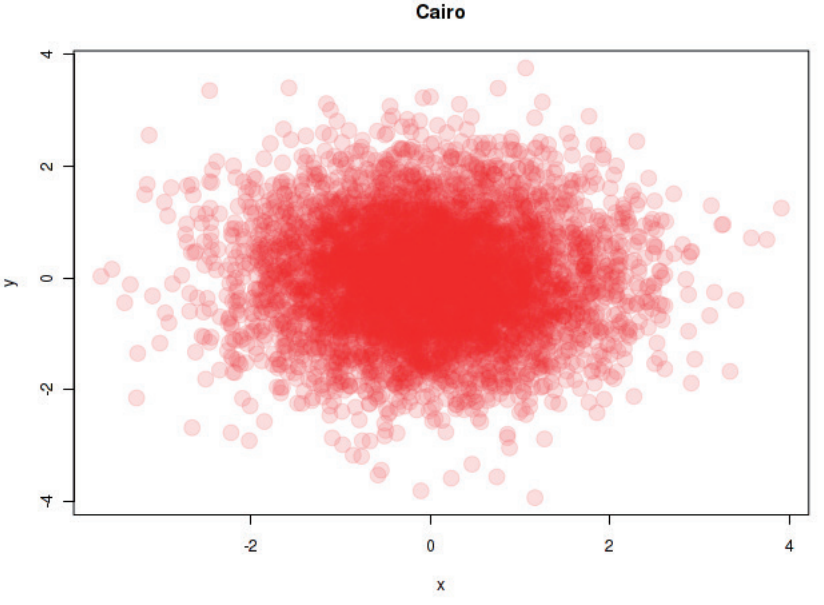


图 1-13　以 CairoPNG() 函数生成的图

SVG 图形输出代码为：

```
> svg(file="plot-svg4.svg",width=6,height=6)
> plot(x,y,col="#ff000018",pch=19,cex=2,main = "plot-svg")
> dev.off()

> CairoSVG(file="Cairo-svg4.svg",width=6,height=6)
> plot(x,y,col="#ff000018",pch=19,cex=2,main = "Cairo-svg")
> dev.off()
```

在当前目录，会生成 2 个 svg 文件，即 plot-svg4.svg 和 Cairo-svg4.svg，可以把 svg 的图片拖拽到浏览器中显示。

2. 三维截面图

接下来，我们再画一个三维截面图，比较函数 png() 和 CairoPNG() 输出的效果。

```
> x <- seq(-10, 10, length= 30)
> y <- x
> f <- function(x,y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
> z <- outer(x, y, f)
> z[is.na(z)] <- 1

> png(file="plot2.png",width=640,height=480)  # PNG 图
> op <- par(bg = "white", mar=c(0,2,3,0)+.1)
> persp(x, y, z,theta = 30, phi = 30,expand = 0.5,col = "lightblue",ltheta =
 120,shade = 0.75,ticktype = "detailed",xlab = "X", ylab = "Y", zlab =
 "Sinc(r)",main = "Plot")
> par(op)
> dev.off()

> CairoPNG(file="Cairo2.png",width=640,height=480)
> op <- par(bg = "white", mar=c(0,2,3,0)+.1)
> persp(x, y, z,theta = 30, phi = 30,expand = 0.5,col = "lightblue",ltheta =
120,shade = 0.75,ticktype = "detailed",xlab = "X", ylab = "Y", zlab =
 "Sinc(r)",main = "Cairo")
> par(op)
> dev.off()
```

在当前目录，会生成 2 个 png 文件，即 plot2.png 和 Cairo2.png，见图 1-14 和图 1-15。

3. 大量文字的图片

最后，我们针对包含大量文字的图片比较 png() 和 CairoPNG() 函数输出的效果。

```
> library(MASS)  # 加载 MASS 包
> data(HairEyeColor)  # 加载 HairEyeColor 数据集
> x <- HairEyeColor[,,1]+HairEyeColor[,,2]
> n <- 100
```

```
> m <- matrix(sample(c(T,F),n^2,replace=T), nr=n, nc=n)
> png(file="plot5.png",width=640,height=480)  # PNG 图
> biplot(corresp(m, nf=2), main="Plot")
> dev.off()

> CairoPNG(file="Cairo5.png",width=640,height=480)
> biplot(corresp(m, nf=2), main="Cairo")
> dev.off()
```

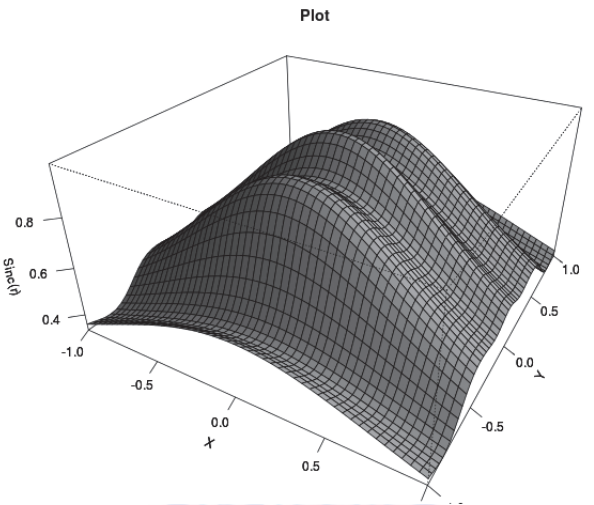


图 1-14 以 png() 函数生成的图

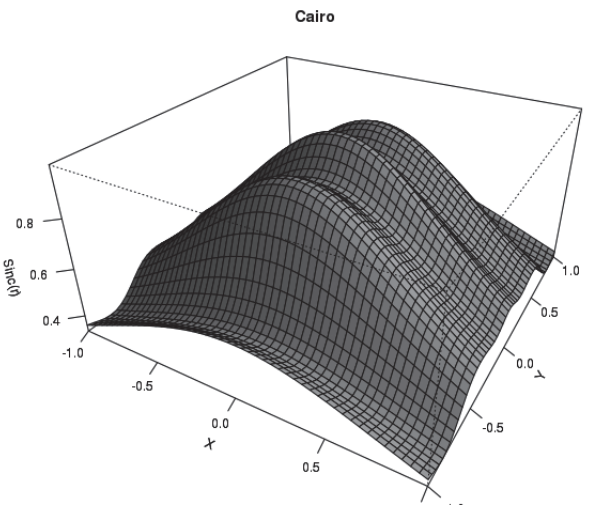


图 1-15 以 CairoPNG() 函数生成的图

在当前目录，会生成 2 个 png 文件，即 plot5.png 和 Cairo5.png，见图 1-16 和图 1-17。

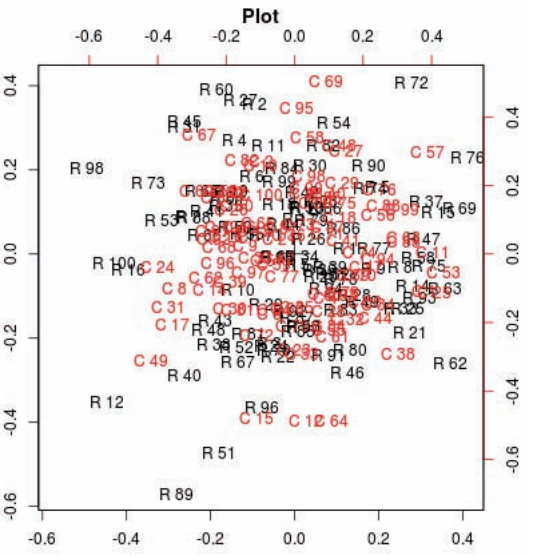


图 1-16　以 png() 函数生成的图

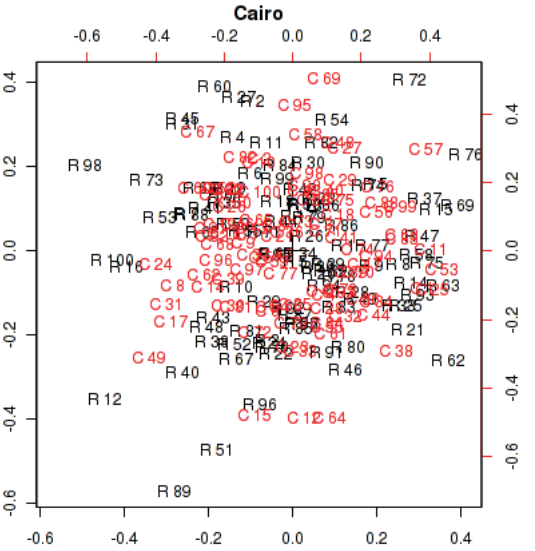


图 1-17　以 CairoPNG() 函数生成的图

我们查看两个文件的属性，发现以 png 直接生成的图 54KB，以 CairoPNG 生成的图 43.8KB，如图 1-18 所示。

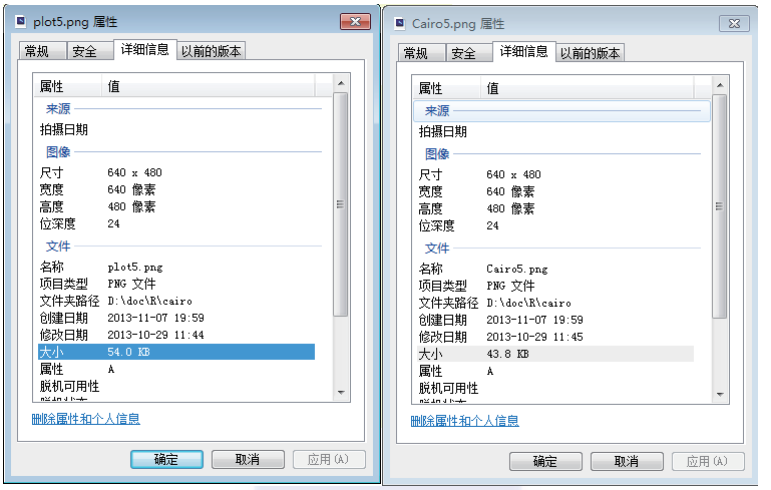


图 1-18 对比两幅图的文件属性

SVG 图形输出代码为：

```
> svg(file="plot-svg5.svg",width=6,height=6)
> biplot(corresp(m, nf=2), main="Plot-svg")
> dev.off()

> CairoSVG(file="Cairo-svg5.svg",width=6,height=6)
> biplot(corresp(m, nf=2), main="Cairo-svg")
> dev.off()
```

在当前目录，会生成 2 个 svg 文件，即 plot-svg5.svg 和 Cairo-svg5.svg。

就以上的 3 个例子来说，我们分辨不出 CairoPNG() 和 png() 之间有太大区别，只是 Cairo 感觉更淡、更柔和一些。关于这一点谢益辉补充说，看不出区别是因为现在 R 的 png() 设备本来就是默认用 cairo，几年前 png() 不用 cairo，所以有时候 PNG 图的质量很差。多数情况下是几乎没有区别，只有少数情况下抗锯齿的表现不一样。

1.8 caTools：一个奇特的工具集

问题

R 除了统计，还能干什么？

引言

R 语言生来就是自由的，不像 Java 和 PHP 等有统一的规范约束。R 语言不仅命名、语法各包各异，就连功能也是各种混搭。caTools 库就是这种混搭库，包括了不相关的几组函数工具集，有图片处理的，有编解码的，有分类器的，有向量计算的，有科学计算的，而且都很好用！以至于我都不知道如何用简短的语言去描述这个包了！唯有用“奇特”来概括它的特点。

1.8.1　caTools 介绍

caTools 是一个基础的工具包，包括移动窗口 (Moving Window) 统计，二进制图片读写，快速计算曲线的面积 AUC，LogitBoost 分类器，base64 的编码器 / 解码器，快速计算舍入、误差、求和、累计求和等函数。下面分别介绍 caTools 包中的 API。

（1）二进制图片读写

❑ read.gif & write.gif: gif 格式图片的读写。

❑ read.ENVI & write.ENVI: ENVI 格式图片的读写，如 GIS 图片。

（2）base64 的编码器 / 解码器 :

❑ base64encode: 编码器。

❑ base64decode: 解码器。

注　Base64 是一种基于 64 个可打印字符来表示二进制数据的表示方法。由于 2 的 6 次方等于 64，所以每 6 个位元为一个单元，对应某个可打印字符。

（3）快速计算曲线的面积 AUC

❑ colAUC: 计算 ROC 曲线的面积 (AUC)。

❑ combs: 向量元素的无序组合。

❑ trapz: 数值积分形梯形法则。

（4）LogitBoost 分类器

❑ LogitBoost: logitBoost 分类算法。

❑ predict.LogitBoost: 预测 logitBoost 分类。

❑ sample.split: 把数据切分成训练集和测试集。

（5）快速计算工具

❑ runmad: 计算向量中位数。

❑ runmean: 计算向量均值。

❑ runmin & runmax: 计算向量最小值和最大值。

❑ runquantile: 计算向量位数。

❑ runsd: 计算向量标准差。

❑ sumexact, cumsumexact: 无误差求和，针对编程语言关于 double 类型的精度优化。

1.8.2 caTools 安装

本节使用的系统环境是：

❑ Linux: Ubuntu Server 12.04.2 LTS 64bit

❑ R: 3.0.1 x86_64-pc-linux-gnu

注 caTools 同时支持 Windows 7 环境和 Linux 环境。

安装 caTools 包非常简单，只需要一条命令就可以完成。

```
~ R  # 启动 R 程序
> install.packages("caTools")  # 执行 caTools 安装命令
```

1.8.3 caTools 使用

在 R 环境中，加载 caTools 类库：

```
> library(caTools)
```

1. 二进制图片读写 gif

（1）写一个 gif 图片

```
> write.gif(volcano, "volcano.gif", col=terrain.colors, flip=TRUE, scale="always",
 comment="Maunga Whau Volcano")  # 取 datasets::volcano 数据集，写入 volcano.gif
```

（2）读一个 gif 图片到内存，再从内存输出

```
> y = read.gif("volcano.gif", verbose=TRUE, flip=TRUE)  # 读入图片到变量 y
```

```
GIF image header
Global colormap with 256 colors
Comment Extension
Image [61 x 87]: 3585 bytes
GIF Terminator

> attributes(y)  # 查看变量的属性
$names
[1] "image"       "col"         "transparent"   "comment"
> class(y$image)  # 查看图片存储矩阵
[1] "matrix"
> ncol(y$image)  # 列数
[1] 61
> nrow(y$image)  # 行数
[1] 87
> head(y$col,10)  # 颜色表，取前 10 个
 [1] "#00A600FF" "#01A600FF" "#03A700FF" "#04A700FF" "#05A800FF" "#07A800FF" "#08A900FF"
 [8] "#09A900FF" "#0BAA00FF" "#0CAA00FF"
> y$comment  # 查看图片备注
[1] "Maunga Whau Volcano"
> image(y$image, col=y$col, main=y$comment, asp=1)  # 通过 y 变量画图，如图 1-19 所示⊖
```

（3）创建一个 Git 动画

```
> x <- y <- seq(-4*pi, 4*pi, len=200)
> r <- sqrt(outer(x^2, y^2, "+"))
> image = array(0, c(200, 200, 10))
> for(i in 1:10) image[,,i] = cos(r-(2*pi*i/10))/(r^.25)
> write.gif(image, "wave.gif", col="rainbow")
> y = read.gif("wave.gif")
> for(i in 1:10) image(y$image[,,i], col=y$col, breaks=(0:256)-0.5, asp=1)
```

在当前的操作目录，会生成一个 wave.gif 的文件，如图 1-20 所示。动画演示效果参见本书在线资源中的文件 wave.gif。

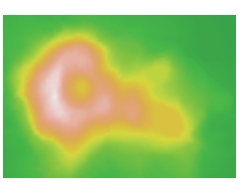


图 1-19　火山地形图

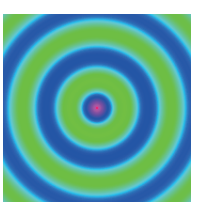


图 1-20　同心圆变化动画

我们看到，caTools 与谢益辉的 animation 包有一样的 GIF 输出动画功能。但使用 caTools 做 gif 动画，不需要依赖其他软件库。而 animation 的 saveGIF 函数需要依赖于

⊖　读者可自己运行代码，查看生成的彩色图片。——编辑注

ImageMagick 或者 GraphicsMagick 等第三方软件。

2. Base64 的编码器 / 解码器

Base64 常用于处理文本数据的场合，表示、传输、存储一些二进制数据，包括 MIME 的 email、email via MIME、在 XML 中存储复杂数据。Base64 的编码解码，只支持向量（vector）类型，不支持 data.frame 和 list 类型。把一个 boolean 向量编解码，代码如下：

```
> size=1  # 设置每个元素占用的字节数
> x = (10*runif(10)>5)
> y = base64encode(x, size=size)
> z = base64decode(y, typeof(x), size=size)

> x  # 原始数据
 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
> y  # 编码后的密文
[1] "AAAAAAEBAAAAAA=="
> z  # 解码后的明文
 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

把一个字符串编解码，代码如下：

```
> x = "Hello R!!" # character
> y = base64encode(x)
> z = base64decode(y, typeof(x))

> x  # 原始数据
[1] "Hello R!!"
> y  # 编码后的密文
[1] "SGVsbG8gUiEh"
> z  # 解码后的明文
[1] "Hello R!!"
```

错误测试：把一个数据框编解码。

```
> data(iris)
> class(iris)
[1] "data.frame"

> head(x)  # 原始数据
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

```
6          5.4         3.9          1.7         0.4  setosa

> base64encode(x)  # 对数据框进行编码
Error in writeBin(x, raw(), size = size, endian = endian) :
  can only write vector objects
```

3. ROC 曲线

ROC 曲线就是接收者操作特征曲线（receiver operating characteristic curve）的简称，这是一种坐标图式的分析工具，主要用于选择最佳的信号侦测模型、舍弃次佳的模型和在同一模型中设定最佳阈值。ROC 曲线首先由二战中的电子工程师和雷达工程师发明，用来侦测战场上的敌军载具（飞机、船舰等），即信号检测，之后很快就被引进心理学来进行信号的知觉检测。数十年来，ROC 分析被用于医学、无线电、生物学、犯罪心理学等领域，而且最近在机器学习（machine learning）和数据挖掘（data mining）领域也得到了很好的发展。

取 MASS::cats 数据集，3 列分别是 Sex（性别）、Bwt（体重）、Hwt（心脏重量）。

```
> library(MASS)
> data(cats)  # 加载数据集
> head(cats)  # 打印前 6 行数据
  Sex Bwt Hwt
1   F 2.0 7.0
2   F 2.0 7.4
3   F 2.0 9.5
4   F 2.1 7.2
5   F 2.1 7.3
6   F 2.1 7.6
> colAUC(cats[,2:3], cats[,1], plotROC=TRUE)  # 计算 ROC 的曲线面积 AUC，输出图片，如图 1-21 所示
              Bwt       Hwt
F vs. M 0.8338451 0.759048
```

从 AUC 判断分类器（预测模型）优劣的标准：

- AUC = 1，是完美分类器，采用这个预测模型时，不管设定什么阈值都能得出完美预测。在绝大多数预测的场合，不存在完美分类器。
- 0.5 < AUC < 1，优于随机猜测。这个分类器（模型）如果妥善设定阈值的话，能有预测价值。
- AUC = 0.5，跟随机猜测一样（例如丢硬币），模型没有预测价值。
- AUC < 0.5，比随机猜测还差；但只要总是反预测而行，就优于随机猜测。

从图 1-21 我们看到 Bwt 和 Hwt 都在 (0.5,1) 之间，因此，cats 的数据集是一个真实有效数据集。如果 cats 的数据集，是一个通过分类预测的数据集，用 AUC 对数据集的评分，就可以检验分类器的好坏了。

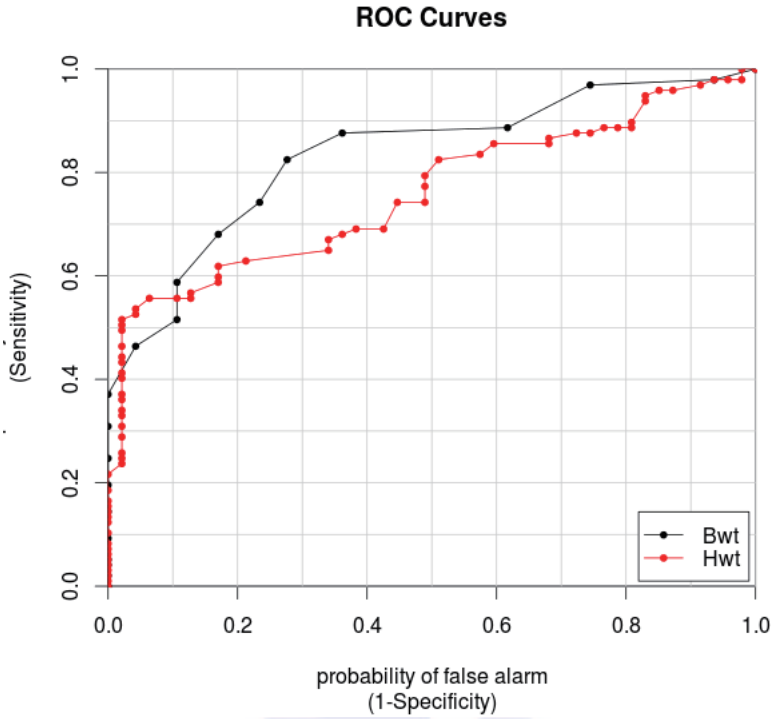


图 1-21 ROC 曲线

4. 向量元素的无序组合

combs(v,k) 函数，用于创建无序的组合的矩阵，矩阵的列表示以几种元素进行组合，矩阵的行表示每种不同的组合。参数 v 是向量；k 是数值，小于等于 v 的长度 [1:length(v)]。

```
> combs(2:5, 3)
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    2    3    5
[3,]    2    4    5
[4,]    3    4    5

> combs(c("cats", "dogs", "mice"), 2)
     [,1]   [,2]
[1,] "cats" "dogs"
[2,] "cats" "mice"
[3,] "dogs" "mice"

> a = combs(1:4, 2)  # 快速组合构建矩阵
> a
```

```
     [,1] [,2]
[1,]    1    2
[2,]    1    3
[3,]    1    4
[4,]    2    3
[5,]    2    4
[6,]    3    4
> b = matrix( c(1,1,1,2,2,3,2,3,4,3,4,4), 6, 2)
> b
     [,1] [,2]
[1,]    1    2
[2,]    1    3
[3,]    1    4
[4,]    2    3
[5,]    2    4
[6,]    3    4
```

5. 数值积分梯形法则

梯形法则原理：将被积函数近似为直线函数，被积的部分近似为梯形。

```
> x = (1:10)*pi/10
> trapz(x, sin(x))
[1] 1.934983

> x = (1:1000)*pi/1000
> trapz(x, sin(x))
[1] 1.999993
```

6. LogitBoost 分类器

取 datasets::iris 数据集，5 列分别是 Sepal.Length(花萼长)、Sepal.Width((花萼宽)、Petal.Length(花瓣长)、Petal.Width(花瓣宽)、Species(种属)。

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

> Data = iris[,-5]
> Label = iris[, 5]
> model = LogitBoost(Data, Label, nIter=20)  # 训练模型
```

```
> model  #模型数据
$Stump
      feature threshhold sign feature threshhold sign feature threshhold sign
 [1,]       3        1.9   -1       2        2.9   -1       4        1.6    1
 [2,]       4        0.6   -1       3        4.7   -1       3        4.8    1
 [3,]       3        1.9   -1       2        2.0   -1       4        1.7    1
 [4,]       4        0.6   -1       3        1.9    1       3        4.9    1
 [5,]       3        1.9   -1       4        1.6   -1       4        1.3    1
 [6,]       4        0.6   -1       1        6.5    1       2        2.6   -1
 [7,]       3        1.9   -1       3        1.9    1       4        1.7    1
 [8,]       4        0.6   -1       2        2.0   -1       2        3.0   -1
 [9,]       3        1.9   -1       3        5.0   -1       3        5.0    1
[10,]       4        0.6   -1       2        2.9    1       1        4.9   -1
[11,]       3        1.9   -1       3        1.9    1       3        4.4    1
[12,]       4        0.6   -1       2        2.0   -1       4        1.7    1
[13,]       3        1.9   -1       3        5.1   -1       2        3.1   -1
[14,]       4        0.6   -1       2        2.0   -1       3        5.1    1
[15,]       3        1.9   -1       3        1.9    1       1        6.5   -1
[16,]       4        0.6   -1       4        1.6   -1       3        5.1    1
[17,]       3        1.9   -1       2        3.1    1       2        3.1   -1
[18,]       4        0.6   -1       3        1.9    1       1        4.9   -1
[19,]       3        1.9   -1       2        2.0   -1       4        1.4    1
[20,]       4        0.6   -1       3        5.1   -1       2        2.2   -1

$lablist
[1] setosa     versicolor virginica
Levels: setosa versicolor virginica

attr(,"class")
[1] "LogitBoost"

> Lab = predict(model, Data)  #分类预测，Lab只显示分类的结果，Prob显示各分类的概率
> Prob = predict(model, Data, type="raw")
> t = cbind(Lab, Prob)  #结果合并，打印前6列
> head(t)
     Lab setosa  versicolor    virginica
[1,]   1      1 0.017986210 1.522998e-08
[2,]   1      1 0.002472623 3.353501e-04
[3,]   1      1 0.017986210 8.315280e-07
[4,]   1      1 0.002472623 4.539787e-05
[5,]   1      1 0.017986210 1.522998e-08
[6,]   1      1 0.017986210 1.522998e-08
```

前6条数据，Lab列表示数据属于分类1，即setosa。其他3列setosa、versicolor、virginica，分类表示属于该分类的概率是多少。下面设置迭代次数，比较分类结果与实际数据结果。

```
> table(predict(model, Data, nIter= 2), Label)  #设置迭代次数为2
            Label
             setosa versicolor virginica
  setosa         48          0         0
  versicolor      0         45         1
  virginica       0          3        45

> table(predict(model, Data, nIter=10), Label)  #设置迭代次数为10
            Label
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         0
  virginica       0          1        47

> table(predict(model, Data), Label)    #默认迭代次数，训练时LogitBoost的nIter值
            Label
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         0
  virginica       0          0        48
```

从上面 3 次测试结果可以看出，迭代次数越多模型分类越准确。下面随机划分训练集和测试集，并分类预测。

```
> mask = sample.split(Label)  #随机取训练集
> length(which(mask))  #训练集99条记录
[1] 99
> length(which(!mask))  #测试集51条记录
[1] 51
> model = LogitBoost(Data[mask,], Label[mask], nIter=10)  #训练模型
> table(predict(model, Data[!mask,], nIter=2), Label[!mask])  #分类预测
             setosa versicolor virginica
  setosa         16          0         0
  versicolor      0         15         3
  virginica       0          1        12

> table(predict(model, Data[!mask,]), Label[!mask])
             setosa versicolor virginica
  setosa         17          0         0
  versicolor      0         16         4
  virginica       0          1        13
```

7. 快速计算工具 runmean

均线在股票交易上非常流行，是一种简单、实用的看盘指标。下面我们对时间序列数据取均线，输出结果如图 1-22 所示。

```
> BJsales  # 取 datasets::BJsales 数据集
Time Series:
Start = 1
End = 150
Frequency = 1
  [1] 200.1 199.5 199.4 198.9 199.0 200.2 198.6 200.0 200.3 201.2 201.6 201.5
 [13] 201.5 203.5 204.9 207.1 210.5 210.5 209.8 208.8 209.5 213.2 213.7 215.1
 [25] 218.7 219.8 220.5 223.8 222.8 223.8 221.7 222.3 220.8 219.4 220.1 220.6
> plot(BJsales,col="black", lty=1,lwd=1, main = "Moving Window Means")
> lines(runmean(BJsales, 3), col="red", lty=2, lwd=2)
> lines(runmean(BJsales, 8), col="green", lty=3, lwd=2)
> lines(runmean(BJsales,15), col="blue", lty=4, lwd=2)
> lines(runmean(BJsales,24), col="magenta", lty=5, lwd=2)
> lines(runmean(BJsales,50), col="cyan", lty=6, lwd=2)
```

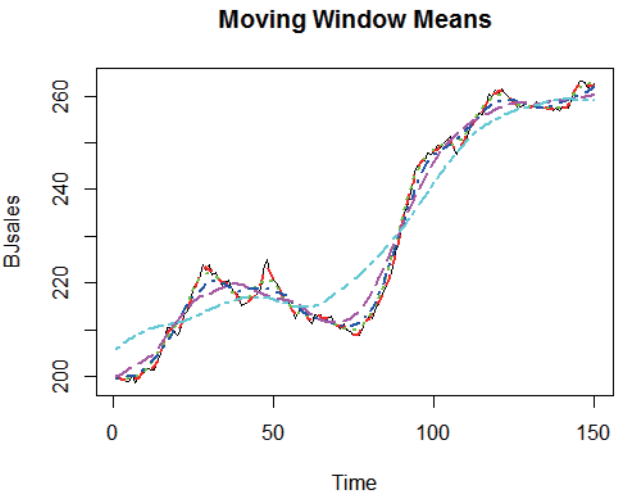


图 1-22 时间序列数据的均线图

我们从图 1-22 中看到 6 条线，黑色是原始数据，其他各种颜色线代表不同单位的均线。比如，红色 (red) 线表示 3 个点的平均，绿色 (green) 线表示 8 个点的平均。⊖

8. 快速计算工具组合

对数据集取最大路径（runmax）、最小路径（runmin）、均值路径（runmean）和中位数路径（runmed），输出结果如图 1-23 所示。

```
> n=200;k=25
> set.seed(100)
> x = rnorm(n,sd=30) + abs(seq(n)-n/4)
```

⊖ 请读者运行源代码来查看彩图。——编辑注

```
> plot(x, main = "Moving Window Analysis Functions (window size=25)")
> lines(runmin (x,k), col="red",lty=1, lwd=1)
> lines(runmax (x,k), col="cyan",lty=1, lwd=1)
> lines(runmean(x,k), col="blue",lty=1, lwd=1)
> lines(runmed (x,k), col="green",lty=2, lwd=2)
```

图 1-23　路径图

9. 无误差求和

各种编程语言，用计算机做小数计算的时候，都会出现计算误差的情况，R 语言也有这个问题。首先用 sum() 函数求和。

```
> x = c(1, 1e20, 1e40, -1e40, -1e20, -1)  # 计算误差求和
> a = sum(x); print(a)
[1] -1e+20
> b = sumexact(x); print(b)  # 无计算误差求和
[1] 0
```

我们对向量 x 求和，各组值加起来正好为 0。但是 sum() 函数，结果是 −1e+20，这是由于编程精度的问题造成的计算误差。通过 sumexact() 函数修正后，就没有误差了。下面用 cumsum() 函数累积求和。

```
> a = cumsum(x); print(a)  # 计算误差累积求和
[1]  1e+00  1e+20  1e+40  0e+00 -1e+20 -1e+20
> b = cumsumexact(x); print(b)  # 无计算误差累积求和
[1] 1e+00 1e+20 1e+40 1e+20 1e+00 0e+00
```

cumsum() 函数同样出现了精度的误差，需要用 cumsumexact() 函数来修正。

最后还是要用“奇特”来概括这个工具集，相信你也发现了它的奇特。

Chapter 2 第 2 章

时间序列基础包

本章主要介绍了时间序列数据处理的 3 个工具包，帮助读者掌握时间序列在 R 语言中的数据结构和基本使用。

2.1 R 语言时间序列基础库 zoo

问题

R 语言怎么处理时间序列数据？

引言

时间序列分析是一种动态数据处理的统计方法，通过对时间序列数据的分析，我们可

以感觉到世界正改变着什么！ R 语言作为统计分析的利器，对时间序列处理有着强大的支持。在 R 语言中，单独为时间序列数据定义了一种数据类型 zoo，zoo 是时间序列的基础，也是股票分析的基础。本节将介绍 zoo 库在 R 语言中的结构和使用。

2.1.1　zoo 包介绍

zoo 是一个 R 语言类库，zoo 类库中定义了一个名为 zoo 的 S3 类型对象，用于描述规则的和不规则的有序的时间序列数据。zoo 对象是一个独立的对象，包括索引、日期、时间，只依赖于基础的 R 环境。zooreg 对象继承了 zoo 对象，只能用于规则的时间序列数据。R 语言中很多其他的程序包，都是以 zoo 和 zooreg 作为时间序列数据的基础的！ zoo 包的 API 主要有 6 类，下面一一介绍。

（1）基础对象

- zoo: 有序的时间序列对象。
- zooreg: 规则的时间序列对象，继承 zoo 对象。与 zoo 相比，不同之处在于 zooreg 要求数据是连续的。

（2）类型转换

- as.zoo: 把一个对象转型为 zoo 类型。
- plot.zoo: 为 plot 函数提供 zoo 的接口。
- xyplot.zoo: 为 lattice 的 xyplot 函数提供 zoo 的接口。
- ggplot2.zoo: 为 ggplot2 包提供 zoo 的接口。

（3）数据操作

- coredata: 查看或编辑 zoo 的数据部分。
- index: 查看或编辑 zoo 的索引部分。
- window.zoo: 按时间过滤数据。
- merge.zoo: 合并多个 zoo 对象。
- read.zoo: 从文件读写 zoo 序列。
- aggregate.zoo: 计算 zoo 数据。
- rollapply: 对 zoo 数据的滚动处理。
- rollmean: 对 zoo 数据的滚动计算均值。

（4）NA 值处理

- na.fill: NA 值的填充。
- na.locf: 替换 NA 值。
- na.aggregate: 计算统计值替换 NA 值。
- na.approx: 计算插值替换 NA 值。

❑ na.StructTS: 计算季节 Kalman 滤波替换 NA 值。

❑ na.trim: 过滤有 NA 的记录。

（5）辅助工具

❑ is.regular: 检查是否是规则的序列。

❑ lag.zoo: 计算步长和差分。

❑ MATCH: 取交集。

❑ ORDER: 值排序，输出索引。

（6）显示控制

❑ yearqtr: 以年季度显示时间。

❑ yearmon: 以年月显示时间。

❑ xblocks: 作图沿 x 轴分割图形。

❑ make.par.list: 用于给 plot.zoo 和 xyplot.zoo 数据格式转换。

2.1.2 zoo 安装

本节使用的系统环境是：

❑ Win7 64bit

❑ R: 3.0.1 x86_64-w64-mingw32/x64 b4bit

注 zoo 同时支持 Windows 7 环境和 Linux 环境。

zoo 包的安装过程如下：

```
~ R  # 启动R程序
> install.packages("zoo")  # 安装zoo包
> library(zoo)  # 加载zoo包
```

2.1.3 zoo 包的使用

1. zoo 对象

zoo 对象包括两部分，即数据部分和索引部分。首先是函数定义：

```
zoo(x = NULL, order.by = index(x), frequency = NULL)
```

其中 x 是数据部分，允许类型为向量、矩阵、因子；order.by 是索引部分，字段唯一性要求，用于排序；frequency 是每个时间单元显示的数量。

以下代码构建一个 zoo 对象，以时间为索引，产生的结果是图 2-1。特别要注意的一点是，zoo 对象可以接受不连续的时间序列数据。

```
> x.Date <- as.Date("2003-02-01") + c(1, 3, 7, 9, 14) - 1  # 定义一个不连续的日期的向量
> x.Date
[1] "2003-02-01" "2003-02-03" "2003-02-07" "2003-02-09" "2003-02-14"
> class(x.Date)
[1] "Date"

> x <- zoo(rnorm(5), x.Date)  # 定义不连续的 zoo 对象
> x
2003-02-01 2003-02-03 2003-02-07 2003-02-09 2003-02-14
0.01964254 0.03122887 0.64721059 1.47397924 1.29109889
> class(x)
[1] "zoo"

> plot(x)  # 画图显示
```

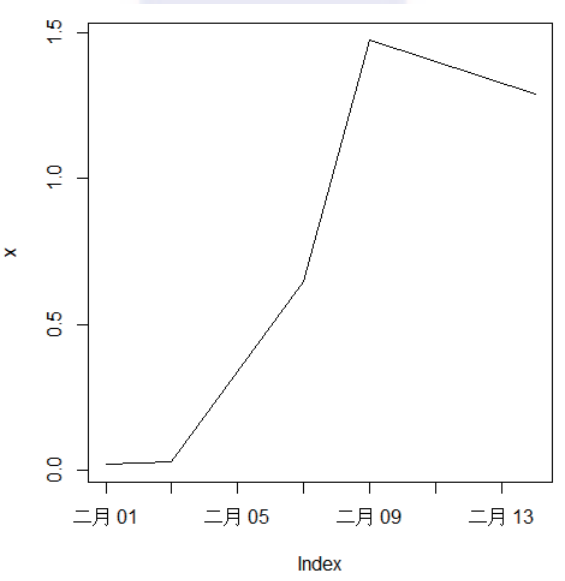


图 2-1　时间序列图

接下来，我们以数字为索引创建多组时间序列。用如下代码，生成一个有 12 个元素的 4 行 3 列的矩阵，以数字 0:10 为索引，创建一个 zoo 类型对象 y，并以图形输出 y，产生的结果如图 2-2 所示。

```
> y <- zoo(matrix(1:12, 4, 3),0:10)
> y
0  1 5  9
1  2 6 10
2  3 7 11
3  4 8 12
```

```
4  1 5  9
5  2 6 10
6  3 7 11
7  4 8 12
8  1 5  9
9  2 6 10
10 3 7 11

> plot(y)  # 矩阵的每一列为一组时间序列图
```

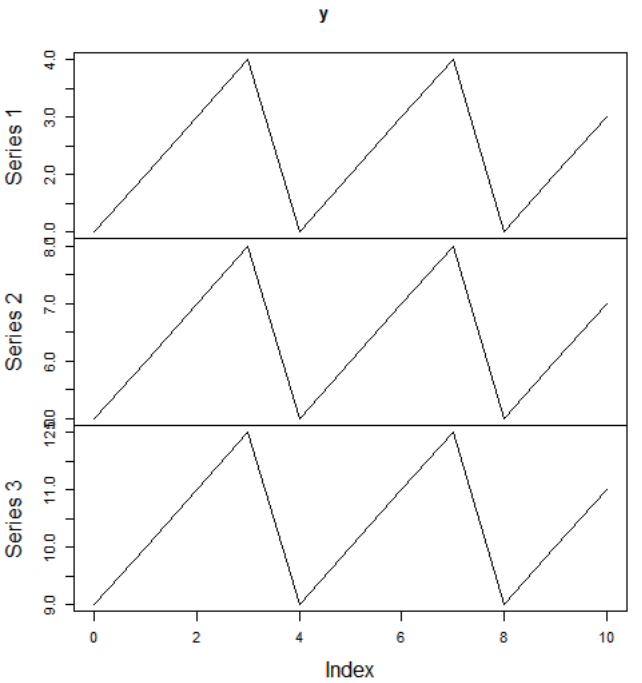


图 2-2 分组的时间序列图

2. zooreg 对象

首先是函数定义：

```
zooreg(data, start = 1, end = numeric(), frequency = 1,
deltat = 1, ts.eps = getOption("ts.eps"), order.by = NULL)
```

下面是参数说明。

- ❑ data: 数据部分，允许类型为向量、矩阵、因子。
- ❑ start: 时间部分，开始时间。
- ❑ end: 时间部分，结束时间。

- frequency: 每个时间单元显示的数量。
- deltat: 连续观测的采样周期，不能与 frequency 同时出现，例如，取每月的数据，为 1/12。
- ts.eps: 时间序列间隔，当数据时间间隔小于 ts.eps 时，使用 ts.eps 作为时间间隔。通过 getOption(“ts.eps”) 设置，默认是 1e-05。
- order.by: 索引部分，字段唯一性要求，用于排序，继承 zoo 的 order.by。

以下代码构建一个 zooreg 对象，以连续的年（季度）时间为索引，产生的结果是图 2-3。

```
> zooreg(1:10, frequency = 4, start = c(1959, 2))
1959(2) 1959(3) 1959(4) 1960(1) 1960(2) 1960(3) 1960(4) 1961(1) 1961(2) 1961(3)
      1       2       3       4       5       6       7       8       9        10

> as.zoo(ts(1:10, frequency = 4, start = c(1959, 2)))
1959(2) 1959(3) 1959(4) 1960(1) 1960(2) 1960(3) 1960(4) 1961(1) 1961(2) 1961(3)
      1       2       3       4       5       6       7       8       9        10

> zr<-zooreg(rnorm(10), frequency = 4, start = c(1959, 2))
> plot(zr)
```

图 2-3　zooreg 对象的时间序列图

3. zoo对象与zooreg对象的区别

zoo对象与zooreg对象的区别体现在计算步长和差分方面。

❑ lag（步长）：zoo根据索引计算，zooreg根据值计算。

❑ diff（差分）：zoo根据索引计算，zooreg根据值计算。

例如，对同一组值不连续的数据(1, 2, 3, 6, 7, 8)，二者的计算结果如下。

```
> x <- c(1, 2, 3, 6, 7, 8)
> zz <- zoo(x, x)
> zr <- as.zooreg(zz)

> lag(zz, k = -1)  # 计算步长

2 3 6 7 8
1 2 3 6 7

> lag(zr, k = -1)
2 3 4 7 8 9
1 2 3 6 7 8

> diff(zz)  # 计算差分
2 3 6 7 8
1 1 3 1 1

> diff(zr)
2 3 7 8
1 1 1 1
```

4. zoo对象的类型转换

首先，把对象从其他类型转型到zoo类型。

```
> as.zoo(rnorm(5))  # 把一个基本类型的向量转型到zoo类型
         1          2          3          4          5
-0.4892119  0.5740950  0.7128003  0.6282868  1.0289573
> as.zoo(ts(rnorm(5), start = 1981, freq = 12))
   1981(1)    1981(2)    1981(3)    1981(4)    1981(5)
 2.3198504  0.5934895 -1.9375893 -1.9888237  1.0944444

> x <- as.zoo(ts(rnorm(5), start = 1981, freq = 12)); x  # 把一个ts类型转型到zoo类型
1981(1)    1981(2)    1981(3)    1981(4)    1981(5)
1.8822996  1.6436364  0.1260436 -2.0360960 -0.1387474
```

其次，把对象从zoo类型转型到其他类型。

```
> as.matrix(x)  # 把zoo类型，转型到矩阵
```

```
                 x
1981(1)  1.8822996
1981(2)  1.6436364
1981(3)  0.1260436
1981(4) -2.0360960
1981(5) -0.1387474

> as.vector(x)  # 把 zoo 类型，转型到数字向量
[1]  1.8822996  1.6436364  0.1260436 -2.0360960 -0.1387474

> as.data.frame(x)  # 把 zoo 类型，转型到数据框
                 x
1981(1)  1.8822996
1981(2)  1.6436364
1981(3)  0.1260436
1981(4) -2.0360960
1981(5) -0.1387474

> as.list(x)  # 把 zoo 类型，转型到列表
[[1]]
   1981(1)    1981(2)    1981(3)    1981(4)    1981(5)
 1.8822996  1.6436364  0.1260436 -2.0360960 -0.1387474
```

5. 用 ggplot2 画时间序列

由于 ggplot2 不支持 zoo 类型的数据，因此需要通过 ggplot2::fortify() 函数，调用 zoo::fortify.zoo() 函数，把 zoo 类型转换成 ggplot2 可识别的类型后，ggplot2 才可以对 zoo 类型数据的画图。以下代码用 ggplot2 画 zoo 类型的时间序列图，产生的结果是图 2-4。

```
> library(ggplot2)  # 加载 ggplot2 包
> library(scales)
> x.Date <- as.Date(paste(2003, 02, c(1, 3, 7, 9, 14), sep = "-"))  # 构建数据对象
> x <- zoo(rnorm(5), x.Date)
> xlow <- x - runif(5)
> xhigh <- x + runif(5)
> z <- cbind(x, xlow, xhigh)
> z  # 显示数据集
                     x        xlow       xhigh
2003-02-01 -0.36006612 -0.88751958 0.006247816
2003-02-03  1.35216617  0.97892538 2.076360524
2003-02-07  0.61920828  0.23746410 1.156569424
2003-02-09  0.27516116  0.09978789 0.777878867
2003-02-14  0.02510778 -0.80107410 0.541592929

# 对 zoo 类型的数据，用 fortify() 转换成 data.frame 类型
```

```
> g<-ggplot(aes(x = Index, y = Value), data = fortify(x, melt = TRUE))
> g<-g+geom_line()
> g<-g+geom_line(aes(x = Index, y = xlow), colour = "red", data = fortify(xlow))
> g<-g+geom_ribbon(aes(x = Index, y = x, ymin = xlow, ymax = xhigh), data =
  fortify(x), fill = "darkgray")
> g<-g+geom_line()
> g<-g+xlab("Index") + ylab("x")
> g
```

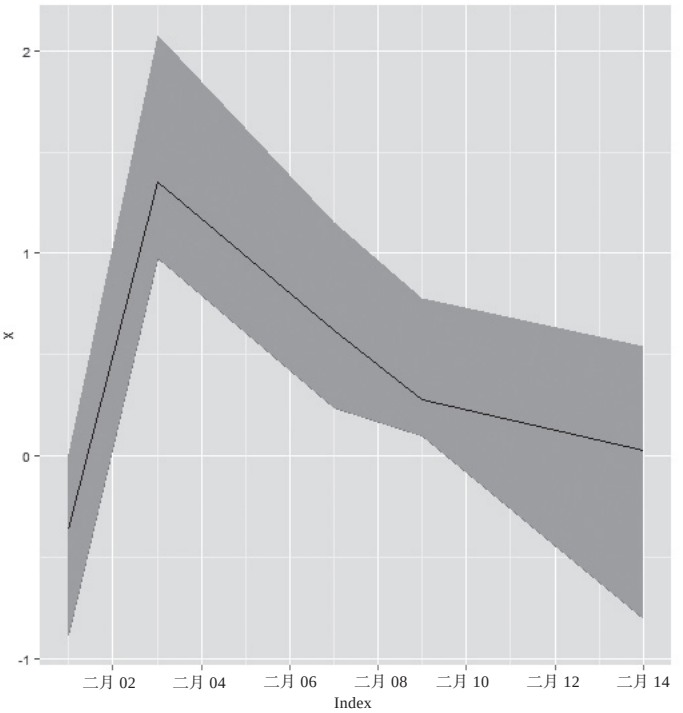


图 2-4 用 ggplot2 画的时间序列图

6. zoo 对象的数据操作

使用 coredata() 函数修改 zoo 类型的数据部分。

```
> x.date <- as.Date(paste(2003, rep(1:4, 4:1), seq(1,20,2), sep = "-"))
> x <- zoo(matrix(rnorm(20), ncol = 2), x.date)
> coredata(x)  # 查看数据部分
            [,1]       [,2]
 [1,] -1.04571765  0.92606273
 [2,] -0.89621126  0.03693769
 [3,]  1.26938716 -1.06620017
```

```
 [4,]  0.59384095 -0.23845635
 [5,]  0.77563432  1.49522344
 [6,]  1.55737038  1.17215855
 [7,] -0.36540180 -1.45770721
 [8,]  0.81655645  0.09505623
 [9,] -0.06063478  0.84766496
[10,] -0.50137832 -1.62436453

> coredata(x) <- matrix(1:20, ncol = 2)  # 修改数据部分
> x  # 查看修改后的数据集
2003-01-01  1 11
2003-01-03  2 12
2003-01-05  3 13
2003-01-07  4 14
2003-02-09  5 15
2003-02-11  6 16
2003-02-13  7 17
2003-03-15  8 18
2003-03-17  9 19
2003-04-19 10 20
```

使用 index() 函数修改 zoo 类型的索引部分。

```
> x.date <- as.Date(paste(2003, rep(1:4, 4:1), seq(1,20,2), sep = "-"))
> x <- zoo(matrix(rnorm(20), ncol = 2), x.date)
> index(x)  # 查看索引部分
 [1] "2003-01-01" "2003-01-03" "2003-01-05" "2003-01-07" "2003-02-09"
 [6] "2003-02-11" "2003-02-13" "2003-03-15" "2003-03-17" "2003-04-19"
> index(x) <- 1:nrow(x)  # 修改索引部分
> index(x)  # 查看修改后的索引部分
 [1]  1  2  3  4  5  6  7  8  9 10
```

使用 window.zoo() 函数按时间过滤数据。

```
> x.date <- as.Date(paste(2003, rep(1:4, 4:1), seq(1,20,2), sep = "-"))
> x <- zoo(matrix(rnorm(20), ncol = 2), x.date)

> window(x, start = as.Date("2003-02-01"), end = as.Date("2003-03-01"))
# 取日期从 2003-02-01 到 2003-03-01 之间的数据
2003-02-09  0.7021167 -0.3073809
2003-02-11  2.5071111  0.6210542
2003-02-13 -1.8900271  0.1819022

> window(x, index = x.date[1:6], start = as.Date("2003-02-01"))
# 取日期从 2003-02-01 开始的，且索引日期在 x.date[1:6] 中的数据
2003-02-09 0.7021167 -0.3073809
2003-02-11 2.5071111  0.6210542
```

```
> window(x, index = x.date[c(4, 8, 10)])  # 取索引日期在x.date[c(4, 8, 10)]中的数据
2003-01-07  1.4623515 -1.198597
2003-03-15 -0.5898128  1.318401
2003-04-19 -0.4209979 -1.648222
```

使用merge.zoo()合并多个zoo对象。

```
> y1 <- zoo(matrix(1:10, ncol = 2), 1:5);y1  # 创建2个zoo数据
1 1  6
2 2  7
3 3  8
4 4  9
5 5 10

> y2 <- zoo(matrix(rnorm(10), ncol = 2), 3:7);y2
3  1.4810127  0.13575871
4 -0.3914258  0.06404148
5  0.6018237  1.85017952
6  1.2964150 -0.12927481
7  0.2211769  0.32381709

> merge(y1, y2, all = FALSE)  # 以相同的索引值合并数据
  y1.1 y1.2       y2.1       y2.2
3    3    8  0.9514985  1.7238941
4    4    9 -1.1131230 -0.2061446
5    5   10  0.6169665 -1.3141951

> merge(y1, y2, all = FALSE, suffixes = c("a", "b"))  # 自定义数据列的名字
  a.1 a.2        b.1        b.2
3   3   8  0.9514985  1.7238941
4   4   9 -1.1131230 -0.2061446
5   5  10  0.6169665 -1.3141951

> merge(y1, y2, all = TRUE)  # 合并完整的数据集，空数据默认以NA填充
  y1.1 y1.2       y2.1       y2.2
1    1    6         NA         NA
2    2    7         NA         NA
3    3    8  0.9514985  1.7238941
4    4    9 -1.1131230 -0.2061446
5    5   10  0.6169665 -1.3141951
6   NA   NA  0.5134937  0.0634741
7   NA   NA  0.3694591 -0.2319775

> merge(y1, y2, all = TRUE, fill = 0)  # 合并完整的数据集，空数据以0填充
  y1.1 y1.2       y2.1       y2.2
1    1    6  0.0000000  0.0000000
2    2    7  0.0000000  0.0000000
```

```
3    3    8  0.9514985  1.7238941
4    4    9 -1.1131230 -0.2061446
5    5   10  0.6169665 -1.3141951
6    0    0  0.5134937  0.0634741
7    0    0  0.3694591 -0.2319775
```

使用 aggregate.zoo() 函数对 zoo 数据进行计算。

```
> x.date <- as.Date(paste(2004, rep(1:4, 4:1), seq(1,20,2), sep = "-"))
# 创建 zoo 类型数据集 x
> x <- zoo(rnorm(12), x.date); x
 2004-01-01  2004-01-03  2004-01-05  2004-01-07  2004-02-09  2004-02-11
 0.67392868  1.95642526 -0.26904101 -1.24455152 -0.39570292  0.09739665
 2004-02-13  2004-03-15  2004-03-17  2004-04-19
-0.23838695 -0.41182796 -1.57721805 -0.79727610

> x.date2 <- as.Date(paste(2004, rep(1:4, 4:1), 1, sep = "-")); x.date2
# 创建时间向量 x.date2
 [1] "2004-01-01" "2004-01-01" "2004-01-01" "2004-01-01" "2004-02-01"
 [6] "2004-02-01" "2004-02-01" "2004-03-01" "2004-03-01" "2004-04-01"

> x2 <- aggregate(x, x.date2, mean); x2  # 计算 x 以 x.date2 为时间分割规则的均值
2004-01-01 2004-02-01 2004-03-01 2004-04-01
 0.2791904 -0.1788977 -0.9945230 -0.7972761
```

7. zoo 对象数据函数化处理

使用 rollapply() 函数对 zoo 数据进行函数化处理。

```
> z <- zoo(11:15, as.Date(31:35))

> rollapply(z, 2, mean)  # 从起始日开始，计算连续 2 日的均值
1970-02-01 1970-02-02 1970-02-03 1970-02-04
      11.5       12.5       13.5       14.5
> rollapply(z, 3, mean)  # 从起始日开始，计算连续 3 日的均值
1970-02-02 1970-02-03 1970-02-04
        12         13         14
```

等价操作变换：用 rollapply() 实现 aggregate() 的操作。

```
> z2 <- zoo(rnorm(6))
> rollapply(z2, 3, mean, by = 3) # means of nonoverlapping groups of 3
         2          5
-0.3065197  0.6350963
> aggregate(z2, c(3,3,3,6,6,6), mean) # same
         3          6
-0.3065197  0.6350963
```

等价操作变换：用 rollapply() 实现 rollmean() 的操作。

```
> rollapply(z2, 3, mean) # uses rollmean which is optimized for mean
         2          3          4          5
-0.3065197 -0.7035811 -0.1672344  0.6350963
> rollmean(z2, 3) # same
         2          3          4          5
-0.3065197 -0.7035811 -0.1672344  0.6350963
```

8. NA 值处理

使用 na.fill() 函数进行 NA 填充。

```
> z <- zoo(c(NA, 2, NA, 3, 4, 5, 9, NA));z  # 创建有 NA 值的 zoo 对象
1  2  3  4  5  6  7  8
NA  2 NA  3  4  5  9 NA

> na.fill(z, "extend")  # 用 extend 的方法，填充 NA 值，即 NA 前后项的均值填充
  1   2   3   4   5   6   7   8
2.0 2.0 2.5 3.0 4.0 5.0 9.0 9.0

> na.fill(z, -(1:3))    # 自定义填充 NA 值，即 -(1:3) 循环填充
 1  2  3  4  5  6  7  8
-1  2 -2  3  4  5  9 -3

> na.fill(z, c("extend", NA))  # 用 extend，配合自定义的方法，即 extend 和自定义规则循环填充
 1  2  3  4  5  6  7  8
 2  2 NA  3  4  5  9  9
```

使用 na.locf() 函数进行 NA 替换。

```
> z <- zoo(c(NA, 2, NA, 3, 4, 5, 9, NA, 11));z
 1  2  3  4  5  6  7  8  9
NA  2 NA  3  4  5  9 NA 11

> na.locf(z)  # 用 NA 的前一项的值，替换 NA 值
 2  3  4  5  6  7  8  9
 2  2  3  4  5  9  9 11

> na.locf(z, fromLast = TRUE)  # 用 NA 的后一项的值，替换 NA 值
 1  2  3  4  5  6  7  8  9
 2  2  3  3  4  5  9 11 11
```

使用 na.aggregate() 函数的统计计算的值替换 NA 值。

```
> z <- zoo(c(1, NA, 3:9),c(as.Date("2010-01-01") + 0:2,as.Date("2010-02-01") +
 0:2,as.Date("2011-01-01") + 0:2));z
```

```
2010-01-01 2010-01-02 2010-01-03 2010-02-01 2010-02-02 2010-02-03 2011-01-01
         1         NA          3          4          5          6          7
2011-01-02 2011-01-03
         8          9

> na.aggregate(z)   # 计算排除 NA 的其他项的均值，替换 NA 值
2010-01-01 2010-01-02 2010-01-03 2010-02-01 2010-02-02 2010-02-03 2011-01-01
     1.000      5.375      3.000      4.000      5.000      6.000      7.000
2011-01-02 2011-01-03
     8.000      9.000

> na.aggregate(z, as.yearmon)    # 以索引的年月分组的均值，替换 NA 值
2010-01-01 2010-01-02 2010-01-03 2010-02-01 2010-02-02 2010-02-03 2011-01-01
         1          2          3          4          5          6          7
2011-01-02 2011-01-03
         8          9

> na.aggregate(z, months)   # 以索引的月份分组的均值，替换 NA 值
2010-01-01 2010-01-02 2010-01-03 2010-02-01 2010-02-02 2010-02-03 2011-01-01
       1.0        5.6        3.0        4.0        5.0        6.0        7.0
2011-01-02 2011-01-03
       8.0        9.0

> na.aggregate(z, format, "%Y")  # 以正则表示的索引的年份分组的均值，替换 NA 值
2010-01-01 2010-01-02 2010-01-03 2010-02-01 2010-02-02 2010-02-03 2011-01-01
       1.0        3.8        3.0        4.0        5.0        6.0        7.0
2011-01-02 2011-01-03
       8.0        9.0
```

使用 na.approx() 函数计算插值替换 NA 值。

```
> z <- zoo(c(2, NA, 1, 4, 5, 2), c(1, 3, 4, 6, 7, 8));z
 1  3  4  6  7  8
 2 NA  1  4  5  2

> na.approx(z)
       1        3        4        6        7        8
2.000000 1.333333 1.000000 4.000000 5.000000 2.000000

> na.approx(z, 1:6)
  1   3   4   6   7   8
2.0 1.5 1.0 4.0 5.0 2.0
```

使用 na.StructTS() 函数计算季节 Kalman 滤波替换 NA 值，产生的结果是图 2-5。

```
> z <- zooreg(rep(10 * seq(4), each = 4) + rep(c(3, 1, 2, 4), times = 4),
            start = as.yearqtr(2000), freq = 4)
```

```
> z[10] <- NA
> zout <- na.StructTS(z);zout
> plot(cbind(z, zout), screen = 1, col = 1:2, type = c("l", "p"), pch = 20)
```

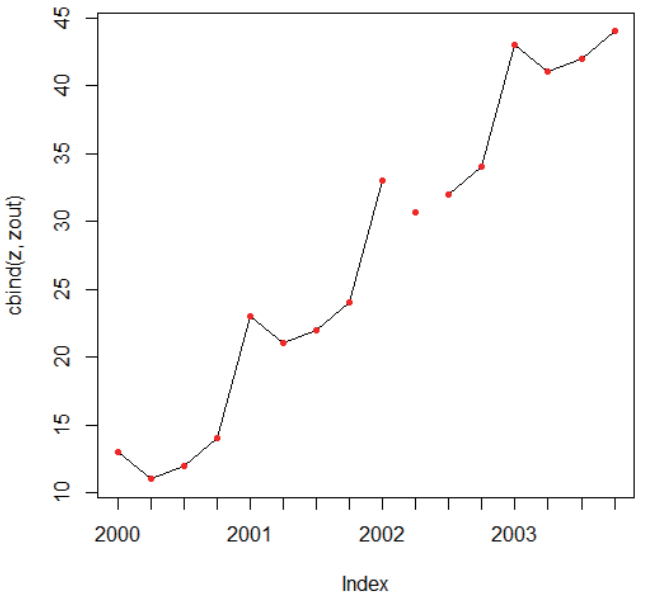


图 2-5 替换 NA 值的时间序列

使用 na.trim() 函数，去掉有 NA 的行。

```
> xx <- zoo(matrix(c(1, 4, 6, NA, NA, 7), 3), c(2, 4, 6));xx
2 1 NA
4 4 NA
6 6  7
> na.trim(xx)
6 6 7
```

9. 数据显示格式

以“年 + 季度”格式输出

```
> x <- as.yearqtr(2000 + seq(0, 7)/4);x  # 以年季默认格式输出
[1] "2000 Q1" "2000 Q2" "2000 Q3" "2000 Q4" "2001 Q1" "2001 Q2" "2001 Q3"
[8] "2001 Q4"
> format(x, "%Y Quarter %q")  # 以年季自定义格式输出
[1] "2000 Quarter 1" "2000 Quarter 2" "2000 Quarter 3" "2000 Quarter 4"
[5] "2001 Quarter 1" "2001 Quarter 2" "2001 Quarter 3" "2001 Quarter 4"
> as.yearqtr("2001 Q2")
```

```
[1] "2001 Q2"
> as.yearqtr("2001 q2")
[1] "2001 Q2"
> as.yearqtr("2001-2")
[1] "2001 Q2"
```

以“年 + 月份”格式输出

```
> x <- as.yearmon(2000 + seq(0, 23)/12) ;x # 以年月默认格式输出
[1] "一月 2000"   "二月 2000"   "三月 2000"   "四月 2000"   "五月 2000"
[6] "六月 2000"   "七月 2000"   "八月 2000"   "九月 2000"   "十月 2000"
[11] "十一月 2000" "十二月 2000" "一月 2001"   "二月 2001"   "三月 2001"
[16] "四月 2001"   "五月 2001"   "六月 2001"   "七月 2001"   "八月 2001"
[21] "九月 2001"   "十月 2001"   "十一月 2001" "十二月 2001"

> as.yearmon("mar07", "%b%y")
[1] NA
> as.yearmon("2007-03-01")
[1] "三月 2007"
> as.yearmon("2007-12")
[1] "十二月 2007"
```

10. 区间分割

使用 xblock() 函数，以不同的颜色划分 3 个区间，即 (-Inf,15)、[15,30] 和 (30,Inf)，产生的是图 2-6。

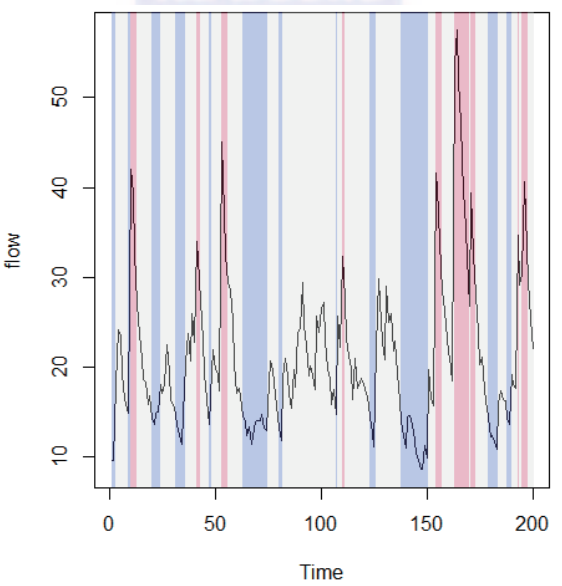


图 2-6　有分割线的时间序列

```
> set.seed(0)
> flow <- ts(filter(rlnorm(200, mean = 1), 0.8, method = "r"))
> rgb <- hcl(c(0, 0, 260), c = c(100, 0, 100), l = c(50, 90, 50), alpha = 0.3)
> plot(flow)
> xblocks(flow > 30, col = rgb[1]) ## high values red
> xblocks(flow < 15, col = rgb[3]) ## low value blue
> xblocks(flow >= 15 & flow <= 30, col = rgb[2]) ## the rest gray
```

11. 从文件读入时间序列数据创建 zoo 对象

我们首先创建一个文件，并将其命名为 read.csv，代码如下。

```
~ vi read.csv
2003-01-01,1.0073644,0.05579711
2003-01-03,-0.2731580,0.06797239
2003-01-05,-1.3096795,-0.20196174
2003-01-07,0.2225738,-1.15801525
2003-02-09,1.1134332,-0.59274327
2003-02-11,0.8373944,0.76606538
2003-02-13,0.3145168,0.03892812
2003-03-15,0.2222181,0.01464681
2003-03-17,-0.8436154,-0.18631697
2003-04-19,0.4438053,1.40059083
```

然后读入文件并生成 zoo 序列。

```
> r <- read.zoo(file="read.csv",sep = ",", format = "%Y-%m-%d")  # 以 zoo 格式读入数据
> r  # 查看数据
                   V2          V3
2003-01-01  1.0073644  0.05579711
2003-01-03 -0.2731580  0.06797239
2003-01-05 -1.3096795 -0.20196174
2003-01-07  0.2225738 -1.15801525
2003-02-09  1.1134332 -0.59274327
2003-02-11  0.8373944  0.76606538
2003-02-13  0.3145168  0.03892812
2003-03-15  0.2222181  0.01464681
2003-03-17 -0.8436154 -0.18631697
2003-04-19  0.4438053  1.40059083
> class(r)  # 查看数据类型
[1] "zoo"
```

我们已经完全掌握了 zoo 库及 zoo 对象的使用，接下来就可以放手去用 R 处理时间序列数据了！

2.2　可扩展的时间序列 xts

问题

如何进行复杂的时间序列数据处理？

引言

本节将继续 2.1 节，介绍 zoo 包的扩展实现。看上去简单的时间序列，却内含复杂的规律。zoo 作为时间序列的基础库，是面向通用的设计，可以用来定义股票数据，也可以分析天气数据。但由于业务行为的不同，我们需要更多的辅助函数帮助我们更高效地完成任务。xts 扩展了 zoo，提供更多的数据处理和数据变换的函数。

2.2.1　xts 介绍

xts 是对时间序列数据 (zoo) 的一种扩展实现，目标是为了统一时间序列的操作接口。实际上，xts 类型继承了 zoo 类型，丰富了时间序列数据处理的函数，API 定义更贴近使用者，更实用，更简单！

1. xts 数据结构

xts 扩展 zoo 的基础结构，由 3 部分组成，如图 2-7 所示。

- ❑ 索引部分：时间类型向量。
- ❑ 数据部分：以矩阵为基础类型，支持可以与矩阵相互转换的任何类型。
- ❑ 属性部分：附件信息，包括时区和索引时间类型的格式等。

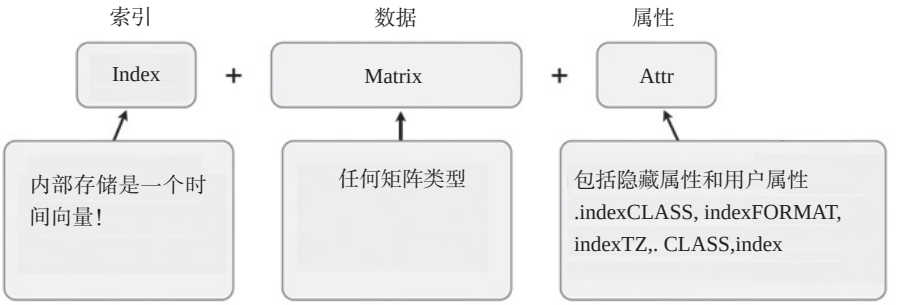


图 2-7 xts 数据结构

2. xts 的 API 介绍

（1）xts 基础

- xts: 定义 xts 数据类型，继承 zoo 类型。
- coredata.xts: 查看或编辑 xts 对象的数据部分。
- xtsAttributes: 查看或编辑 xts 对象的属性部分。
- [.xts]: 用 [] 语法，取数据子集。
- dimnames.xts: 查看或编辑 xts 维度名。
- sample_matrix: 测试数据集，包括 180 条 xts 对象的记录，matrix 类型。
- xtsAPI: C 语言 API 接口。

（2）类型转换

- as.xts: 转换对象到 xts(zoo) 类型。
- as.xts.methods: 转换对象到 xts 函数。
- plot.xts: 为 plot 函数提供 xts 的接口作图。
- .parseISO8601: 把字符串 (ISO8601 格式) 输出为 POSIXct 类型的，包括开始时间和结束时间的 list 对象。
- firstof: 创建一个开始时间，POSIXct 类型。
- lastof: 创建一个结束时间，POSIXct 类型。
- indexClass: 取索引类型。
- .indexDate: 索引的日期。
- .indexday: 索引的日期，同 .indexDate。
- .indexyday: 索引的年 (日) 值。
- .indexmday: 索引的月 (日) 值。
- .indexwday: 索引的周 (日) 值。

- .indexweek: 索引的周值。
- .indexmon: 索引的月值。
- .indexyear: 索引的年值。
- .indexhour: 索引的时值。
- .indexmin: 索引的分值。
- .indexsec: 索引的秒值。

（3）数据处理

- align.time: 以下一个时间对齐数据，秒，分钟，小时。
- endpoints: 按时间单元提取索引数据。
- merge.xts: 合并多个 xts 对象，重写 zoo::merge.zoo 函数。
- rbind.xts: 数据按行合并，为 rbind 函数提供 xts 的接口。
- split.xts: 数据分割，为 split 函数，提供 xts 的接口。
- na.locf.xts: 替换 NA 值，重写 zoo:na.locf 函数。

（4）数据统计

- apply.daily: 按日分割数据，执行函数。
- apply.weekly: 按周分割数据，执行函数。
- apply.monthly: 按月分割数据，执行函数。
- apply.quarterly: 按季分割数据，执行函数。
- apply.yearly: 按年分割数据，执行函数。
- to.period: 按期间分割数据。
- period.apply: 按期间执行自定义函数。
- period.max: 按期间计算最大值。
- period.min: 按期间计算最小值。
- period.prod: 按期间计算指数。
- period.sum: 按期间求和。
- nseconds: 计算数据集包括多少秒。
- nminutes: 计算数据集包括多少分。
- nhours: 计算数据集包括多少时。
- ndays: 计算数据集包括多少日。
- nweeks: 计算数据集包括多少周。
- nmonths: 计算数据集包括多少月。
- nquarters: 计算数据集包括多少季。
- nyears: 计算数据集包括多少年。

❑ periodicity: 查看时间序列的期间。

（5）辅助工具

❑ first: 从开始到结束设置条件取子集。

❑ last: 从结束到开始设置条件取子集。

❑ timeBased: 判断是否是时间类型。

❑ timeBasedSeq: 创建时间的序列。

❑ diff.xts: 计算步长和差分。

❑ isOrdered: 检查向量是否是顺序的。

❑ make.index.unique: 强制时间唯一，增加毫秒随机数。

❑ axTicksByTime: 计算 X 轴刻度标记位置按时间描述。

❑ indexTZ: 查询 xts 对象的时区。

2.2.2 xts 包的安装

本节使用的系统环境是：

❑ Win7 64bit

❑ R: 3.0.1 x86_64-w64-mingw32/x64 b4bit

注 xts 同时支持 Windows 7 环境和 Linux 环境。

xts 的安装过程如下：

```
~ R  # 启动 R 程序
> install.packages("xts")  # 安装 xts 包
also installing the dependency 'zoo'
> library(xts)  # 加载 xts
```

2.2.3 xts 包的使用

1. xts 对象的基本操作

查看 xts 包中的测试数据集 sample_matrix。

```
> data(sample_matrix)  # 加载 sample_matrix 数据集
> head(sample_matrix)  # 查看 sample_matrix 数据集的前 6 条数据
               Open     High      Low    Close
2007-01-02 50.03978 50.11778 49.95041 50.11778
2007-01-03 50.23050 50.42188 50.23050 50.39767
2007-01-04 50.42096 50.42096 50.26414 50.33236
2007-01-05 50.37347 50.37347 50.22103 50.33459
2007-01-06 50.24433 50.24433 50.11121 50.18112
```

```
2007-01-07 50.13211 50.21561 49.99185 49.99185
```

接下来，定义一个 xts 类型对象。

```
> sample.xts <- as.xts(sample_matrix, descr='my new xts object')
                      # 创建一个 xts 对象，并设置属性 descr
> class(sample.xts)  # xts 是继承 zoo 类型的对象
[1] "xts" "zoo"

> str(sample.xts)    # 打印对象结构
An 'xts' object on 2007-01-02/2007-06-30 containing:
  Data: num [1:180, 1:4] 50 50.2 50.4 50.4 50.2 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:4] "Open" "High" "Low" "Close"
  Indexed by objects of class: [POSIXct,POSIXt] TZ:
  xts Attributes:
List of 1
 $ descr: chr "my new xts object"

> attr(sample.xts,'descr')  # 查看对象的属性 descr
[1] "my new xts object"
```

在 [] 中，通过字符串匹配进行 xts 数据查询。

```
> head(sample.xts['2007'])   # 选出 2007 年的数据
               Open     High      Low    Close
2007-01-02 50.03978 50.11778 49.95041 50.11778
2007-01-03 50.23050 50.42188 50.23050 50.39767
2007-01-04 50.42096 50.42096 50.26414 50.33236
2007-01-05 50.37347 50.37347 50.22103 50.33459
2007-01-06 50.24433 50.24433 50.11121 50.18112
2007-01-07 50.13211 50.21561 49.99185 49.99185

> head(sample.xts['2007-03/'])  # 选出 2007 年 03 月的数据
               Open     High      Low    Close
2007-03-01 50.81620 50.81620 50.56451 50.57075
2007-03-02 50.60980 50.72061 50.50808 50.61559
2007-03-03 50.73241 50.73241 50.40929 50.41033
2007-03-04 50.39273 50.40881 50.24922 50.32636
2007-03-05 50.26501 50.34050 50.26501 50.29567
2007-03-06 50.27464 50.32019 50.16380 50.16380

> head(sample.xts['2007-03-06/2007'])  # 选出 2007 年 03 月 06 日到 2007 年的数据
               Open     High      Low    Close
2007-03-06 50.27464 50.32019 50.16380 50.16380
2007-03-07 50.14458 50.20278 49.91381 49.91381
```

```
2007-03-08 49.93149 50.00364 49.84893 49.91839
2007-03-09 49.92377 49.92377 49.74242 49.80712
2007-03-10 49.79370 49.88984 49.70385 49.88698
2007-03-11 49.83062 49.88295 49.76031 49.78806

> sample.xts['2007-01-03']  # 选出 2007 年 01 月 03 日的数据
              Open     High     Low    Close
2007-01-03 50.2305 50.42188 50.2305 50.39767
```

2. 用 xts 对象画图

用 xts 对象可以画曲线图（图 2-8）和 K 线图（图 2-9），下面是产生这两种图的代码，首先是曲线图：

```
> data(sample_matrix)
> plot(as.xts(sample_matrix))
Warning message:
In plot.xts(as.xts(sample_matrix)) :
  only the univariate series will be plotted
```

警告信息提示，只有单变量序列将被绘制，即只画出第一列数据 sample_matrix[,1] 的曲线。

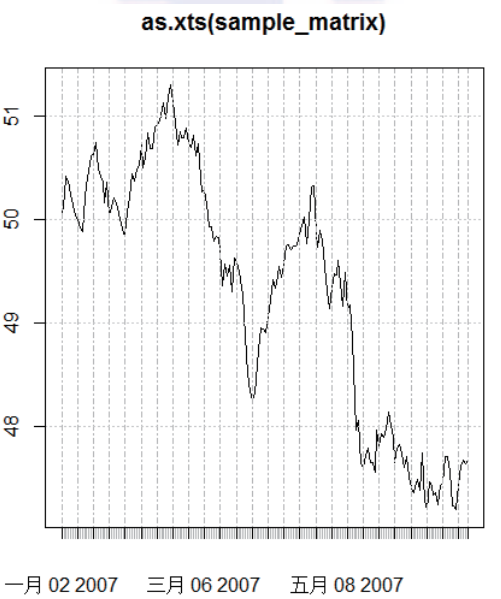


图 2-8 曲线图

然后是 K 线图：

```
> plot(as.xts(sample_matrix), type='candles')  # 画 K 线图
```

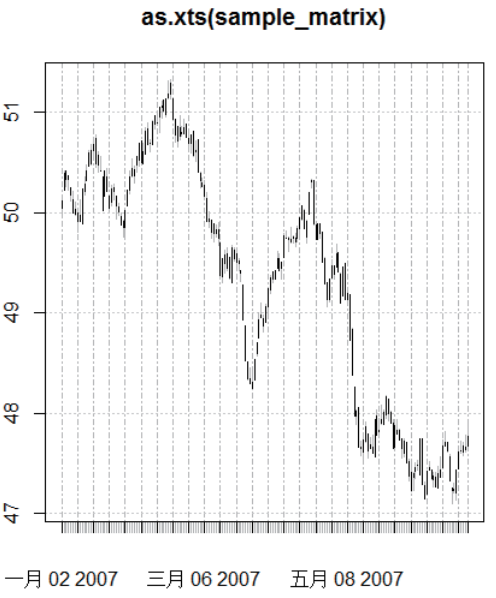


图 2-9　K 线图

3. xts 对象的类型转换

创建首尾时间函数 firstof() 和 lastof()。

```
> firstof(2000)  # 2000 年的第一天，时分秒显示省略
[1] "2000-01-01 CST"
> firstof(2005,01,01)
[1] "2005-01-01 CST"
> lastof(2007)  # 2007 年的最后一天，最后一秒
[1] "2007-12-31 23:59:59.99998 CST"
> lastof(2007,10)
[1] "2007-10-31 23:59:59.99998 CST"
```

创建首尾时间。

```
> .parseISO8601('2000')  # 以 ISO8601 格式，创建 2000 年首尾时间
$first.time
[1] "2000-01-01 CST"
$last.time
[1] "2000-12-31 23:59:59.99998 CST"

> .parseISO8601('2000-05/2001-02')
```

```
# 以ISO8601格式，创建2000年05月开始，2001年02月结束的时间
$first.time
[1] "2000-05-01 CST"
$last.time
[1] "2001-02-28 23:59:59.99998 CST"

> .parseISO8601('2000-01/02')
$first.time
[1] "2000-01-01 CST"
$last.time
[1] "2000-02-29 23:59:59.99998 CST"

> .parseISO8601('T08:30/T15:00')
$first.time
[1] "1970-01-01 08:30:00 CST"
$last.time
[1] "1970-12-31 15:00:59.99999 CST"
```

创建以时间类型为索引的 xts 对象。

```
> x <- timeBasedSeq('2010-01-01/2010-01-02 12:00')  # 创建POSIXt类型时间
> head(x)
[1] "2010-01-01 00:00:00 CST"
[2] "2010-01-01 00:01:00 CST"
[3] "2010-01-01 00:02:00 CST"
[4] "2010-01-01 00:03:00 CST"
[5] "2010-01-01 00:04:00 CST"
[6] "2010-01-01 00:05:00 CST"
> class(x)
[1] "POSIXt"  "POSIXct"
> x <- xts(1:length(x), x)   # 以时间为索引创建xts对象
> head(x)
                    [,1]
2010-01-01 00:00:00    1
2010-01-01 00:01:00    2
2010-01-01 00:02:00    3
2010-01-01 00:03:00    4
2010-01-01 00:04:00    5
2010-01-01 00:05:00    6
> indexClass(x)
[1] "POSIXt"  "POSIXct"
```

格式化索引时间的显示。

```
> indexFormat(x) <- "%Y-%b-%d %H:%M:%OS3"  # 通过正则格式化索引的时间显示
> head(x)
                          [,1]
```

```
2010-一月-01 00:00:00.000    1
2010-一月-01 00:01:00.000    2
2010-一月-01 00:02:00.000    3
2010-一月-01 00:03:00.000    4
2010-一月-01 00:04:00.000    5
2010-一月-01 00:05:00.000    6
```

查看索引时间。

```
> .indexhour(head(x))  # 按小时取索引时间
[1] 0 0 0 0 0 0
> .indexmin(head(x))   # 按分钟取索引时间
[1] 0 1 2 3 4 5
```

4. xts 对象的数据处理

数据对齐。

```
> x <- Sys.time() + 1:30
> align.time(x, 10)    # 整 10 秒对齐，秒位为 10 的整数倍
 [1] "2013-11-18 15:42:30 CST" "2013-11-18 15:42:30 CST"
 [3] "2013-11-18 15:42:30 CST" "2013-11-18 15:42:40 CST"
 [5] "2013-11-18 15:42:40 CST" "2013-11-18 15:42:40 CST"
 [7] "2013-11-18 15:42:40 CST" "2013-11-18 15:42:40 CST"
 [29] "2013-11-18 15:43:00 CST" "2013-11-18 15:43:00 CST"

> align.time(x, 60)    # 整 60 秒对齐，秒位为 0，分位为整数
 [1] "2013-11-18 15:43:00 CST" "2013-11-18 15:43:00 CST"
 [3] "2013-11-18 15:43:00 CST" "2013-11-18 15:43:00 CST"
 [5] "2013-11-18 15:43:00 CST" "2013-11-18 15:43:00 CST"
 [7] "2013-11-18 15:43:00 CST" "2013-11-18 15:43:00 CST"
 [9] "2013-11-18 15:43:00 CST" "2013-11-18 15:43:00 CST"
[11] "2013-11-18 15:43:00 CST" "2013-11-18 15:43:00 CST"
```

按时间分割数据，并计算。

```
> xts.ts <- xts(rnorm(231),as.Date(13514:13744,origin="1970-01-01"))
> apply.monthly(xts.ts,mean)  # 按月计算均值，以每月的最后一日显示
                 [,1]
2007-01-31  0.17699984
2007-02-28  0.30734220
2007-03-31 -0.08757189
2007-04-30  0.18734688
2007-05-31  0.04496954
2007-06-30  0.06884836
2007-07-31  0.25081814
2007-08-19 -0.28845938
```

```
> apply.monthly(xts.ts,function(x) var(x))
# 按月计算自定义函数(方差),以每月的最后一日显示
                [,1]
2007-01-31 0.9533217
2007-02-28 0.9158947
2007-03-31 1.2821450
2007-04-30 1.2805976
2007-05-31 0.9725438
2007-06-30 1.5228904
2007-07-31 0.8737030
2007-08-19 0.8490521

> apply.quarterly(xts.ts,mean)  # 按季计算均值,以每季的最后一日显示
                 [,1]
2007-03-31 0.12642053
2007-06-30 0.09977926
2007-08-19 0.04589268

> apply.yearly(xts.ts,mean)  # 按年计算均值,以年季的最后一日显示
                 [,1]
2007-08-19 0.09849522
```

使用 to.period() 函数按间隔分割数据。

```
> data(sample_matrix)
> to.period(sample_matrix)  # 默认按月分割矩阵数据
           sample_matrix.Open sample_matrix.High sample_matrix.Low sample_matrix.Close
2007-01-31           50.03978           50.77336          49.76308            50.22578
2007-02-28           50.22448           51.32342          50.19101            50.77091
2007-03-31           50.81620           50.81620          48.23648            48.97490
2007-04-30           48.94407           50.33781          48.80962            49.33974
2007-05-31           49.34572           49.69097          47.51796            47.73780
2007-06-30           47.74432           47.94127          47.09144            47.76719
> class(to.period(sample_matrix))
[1] "matrix"

> samplexts <- as.xts(sample_matrix)  # 默认按月分割xts类型数据
> to.period(samplexts)
           samplexts.Open samplexts.High samplexts.Low samplexts.Close
2007-01-31       50.03978       50.77336      49.76308        50.22578
2007-02-28       50.22448       51.32342      50.19101        50.77091
2007-03-31       50.81620       50.81620      48.23648        48.97490
2007-04-30       48.94407       50.33781      48.80962        49.33974
2007-05-31       49.34572       49.69097      47.51796        47.73780
2007-06-30       47.74432       47.94127      47.09144        47.76719
> class(to.period(samplexts))
[1] "xts" "zoo"
```

使用 endpoints() 函数，按间隔分割索引数据。

```
> data(sample_matrix)
> endpoints(sample_matrix)  # 默认按月分割
[1]   0  30  58  89 119 150 180
> endpoints(sample_matrix, 'days',k=7)  # 按每 7 日分割
 [1]   0   6  13  20  27  34  41  48  55  62  69  76  83  90  97 104 111 118 125
[20] 132 139 146 153 160 167 174 180

> endpoints(sample_matrix, 'weeks')     # 按周分割
 [1]   0   7  14  21  28  35  42  49  56  63  70  77  84  91  98 105 112 119 126
[20] 133 140 147 154 161 168 175 180

> endpoints(sample_matrix, 'months')    # 按月分割
[1]   0  30  58  89 119 150 180
```

使用 merge() 函数进行数据合并，按列合并。

```
> (x <- xts(4:10, Sys.Date()+4:10))     # 创建 2 个 xts 数据集
           [,1]
2013-11-22    4
2013-11-23    5
2013-11-24    6
2013-11-25    7
2013-11-26    8
2013-11-27    9
2013-11-28   10
> (y <- xts(1:6, Sys.Date()+1:6))
           [,1]
2013-11-19    1
2013-11-20    2
2013-11-21    3
2013-11-22    4
2013-11-23    5
2013-11-24    6

> merge(x,y)  # 按列合并数据，空项以 NA 填空
            x  y
2013-11-19 NA  1
2013-11-20 NA  2
2013-11-21 NA  3
2013-11-22  4  4
2013-11-23  5  5
2013-11-24  6  6
2013-11-25  7 NA
2013-11-26  8 NA
2013-11-27  9 NA
```

```
2013-11-28 10 NA

> merge(x,y, join='inner')  #按索引合并数据
           x y
2013-11-22 4 4
2013-11-23 5 5
2013-11-24 6 6

> merge(x,y, join='left')  #以左侧为基础合并数据
            x  y
2013-11-22  4  4
2013-11-23  5  5
2013-11-24  6  6
2013-11-25  7 NA
2013-11-26  8 NA
2013-11-27  9 NA
2013-11-28 10 NA
```

使用 rbind() 函数进行数据合并，按行合并。

```
> x <- xts(1:3, Sys.Date()+1:3)
> rbind(x,x)  # 按行合并数据
           [,1]
2013-11-19    1
2013-11-19    1
2013-11-20    2
2013-11-20    2
2013-11-21    3
2013-11-21    3
```

使用 split() 函数进行数据切片，按行切片。

```
> data(sample_matrix)
> x <- as.xts(sample_matrix)

> split(x)[[1]]  # 默认按月进行切片，打印第一个月的数据
               Open     High      Low    Close
2007-01-02 50.03978 50.11778 49.95041 50.11778
2007-01-03 50.23050 50.42188 50.23050 50.39767
2007-01-04 50.42096 50.42096 50.26414 50.33236
2007-01-05 50.37347 50.37347 50.22103 50.33459
2007-01-06 50.24433 50.24433 50.11121 50.18112
2007-01-07 50.13211 50.21561 49.99185 49.99185
2007-01-08 50.03555 50.10363 49.96971 49.98806

> split(x, f="weeks")[[1]]  # 按周切片，打印前1周数据
               Open     High      Low    Close
```

```
2007-01-02 50.03978 50.11778 49.95041 50.11778
2007-01-03 50.23050 50.42188 50.23050 50.39767
2007-01-04 50.42096 50.42096 50.26414 50.33236
2007-01-05 50.37347 50.37347 50.22103 50.33459
2007-01-06 50.24433 50.24433 50.11121 50.18112
2007-01-07 50.13211 50.21561 49.99185 49.99185
2007-01-08 50.03555 50.10363 49.96971 49.98806
```

NA 值处理。

```
> x <- xts(1:10, Sys.Date()+1:10)
> x[c(1,2,5,9,10)] <- NA
> x
           [,1]
2013-11-19   NA
2013-11-20   NA
2013-11-21    3
2013-11-22    4
2013-11-23   NA
2013-11-24    6
2013-11-25    7
2013-11-26    8
2013-11-27   NA
2013-11-28   NA

> na.locf(x)  # 取 NA 的前一个，替换 NA 值
           [,1]
2013-11-19   NA
2013-11-20   NA
2013-11-21    3
2013-11-22    4
2013-11-23    4
2013-11-24    6
2013-11-25    7
2013-11-26    8
2013-11-27    8
2013-11-28    8

> na.locf(x, fromLast=TRUE)  # 取 NA 后一个，替换 NA 值
           [,1]
2013-11-19    3
2013-11-20    3
2013-11-21    3
2013-11-22    4
2013-11-23    6
2013-11-24    6
2013-11-25    7
```

```
2013-11-26    8
2013-11-27   NA
2013-11-28   NA
```

5. xts对象的数据统计计算

对xts对象可以进行各种数据统计计算，比如取开始时间和结束时间，计算时间区间，按期间计算统计指标。

（1）取xts对象的开始时间和结束时间，具体代码如下：

```
> xts.ts <- xts(rnorm(231),as.Date(13514:13744,origin="1970-01-01"))
> start(xts.ts)  # 取开始时间
[1] "2007-01-01"
> end(xts.ts)  # 取结束时间
[1] "2007-08-19"
> periodicity(xts.ts)  # 以日为单位，打印开始和结束时间
Daily periodicity from 2007-01-01 to 2007-08-19
```

（2）计算时间区间函数，具体代码如下：

```
> data(sample_matrix)
> ndays(sample_matrix)  # 计算数据有多少日
[1] 180
> nweeks(sample_matrix)  # 计算数据有多少周
[1] 26
> nmonths(sample_matrix)  # 计算数据有多少月
[1] 6
> nquarters(sample_matrix)  # 计算数据有多少季
[1] 2
> nyears(sample_matrix)  # 计算数据有多少年
[1] 1
```

（3）按期间计算统计指标，具体代码如下：

```
> zoo.data <- zoo(rnorm(31)+10,as.Date(13514:13744,origin="1970-01-01"))
> ep <- endpoints(zoo.data,'weeks')  # 按周获得期间索引
> ep
 [1]   0   7  14  21  28  35  42  49  56  63  70  77  84  91  98 105 112 119
[19] 126 133 140 147 154 161 168 175 182 189 196 203 210 217 224 231

> period.apply(zoo.data, INDEX=ep, FUN=function(x) mean(x))  # 计算周的均值
2007-01-07 2007-01-14 2007-01-21 2007-01-28 2007-02-04 2007-02-11 2007-02-18
 10.200488   9.649387  10.304151   9.864847  10.382943   9.660175   9.857894
2007-02-25 2007-03-04 2007-03-11 2007-03-18 2007-03-25 2007-04-01 2007-04-08
 10.495037   9.569531  10.292899   9.651616  10.089103   9.961048  10.304860
2007-04-15 2007-04-22 2007-04-29 2007-05-06 2007-05-13 2007-05-20 2007-05-27
```

```
  9.658432   9.887531  10.608082   9.747787  10.052955   9.625730  10.430030
2007-06-03 2007-06-10 2007-06-17 2007-06-24 2007-07-01 2007-07-08 2007-07-15
  9.814703  10.224869   9.509881  10.187905  10.229310  10.261725   9.855776
2007-07-22 2007-07-29 2007-08-05 2007-08-12 2007-08-19
  9.445072  10.482020   9.844531  10.200488   9.649387

> head(period.max(zoo.data, INDEX=ep))  # 计算周的最大值
              [,1]
2007-01-07 12.05912
2007-01-14 10.79286
2007-01-21 11.60658
2007-01-28 11.63455
2007-02-04 12.05912
2007-02-11 10.67887

> head(period.min(zoo.data, INDEX=ep))  # 计算周的最小值
              [,1]
2007-01-07 8.874509
2007-01-14 8.534655
2007-01-21 9.069773
2007-01-28 8.461555
2007-02-04 9.421085
2007-02-11 8.534655

> head(period.prod(zoo.data, INDEX=ep)) # 计算周的一个指数值
              [,1]
2007-01-07 11140398
2007-01-14  7582350
2007-01-21 11930334
2007-01-28  8658933
2007-02-04 12702505
2007-02-11  7702767
```

6. xts 对象的时间序列操作

检查时间类型。

```
> class(Sys.time());timeBased(Sys.time())  # Sys.time() 是时间类型 POSIXct
[1] "POSIXct" "POSIXt"
[1] TRUE

> class(Sys.Date());timeBased(Sys.Date())  # Sys.Date() 是时间类型 Date
[1] "Date"
[1] TRUE

> class(20070101);timeBased(20070101)  # 20070101 不是时间类型
```

```
[1] "numeric"
[1] FALSE
```

使用 timeBasedSeq() 函数创建时间序列。

```
> timeBasedSeq('1999/2008')   # 按年
 [1] "1999-01-01" "2000-01-01" "2001-01-01" "2002-01-01" "2003-01-01"
 [6] "2004-01-01" "2005-01-01" "2006-01-01" "2007-01-01" "2008-01-01"

> head(timeBasedSeq('199901/2008'))  # 按月
[1] "十二月 1998" "一月 1999"   "二月 1999"   "三月 1999"   "四月 1999"
[6] "五月 1999"

> head(timeBasedSeq('199901/2008/d'),40)  # 按日
 [1] "十二月 1998" "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"
 [6] "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"
[11] "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"
[16] "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"
[21] "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"
[26] "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"   "一月 1999"
[31] "一月 1999"   "一月 1999"   "二月 1999"   "二月 1999"   "二月 1999"
[36] "二月 1999"   "二月 1999"   "二月 1999"   "二月 1999"   "二月 1999"

> timeBasedSeq('20080101 0830',length=100)  # 按数量创建，100分钟的数据集
$from
[1] "2008-01-01 08:30:00 CST"
$to
[1] NA
$by
[1] "mins"
$length.out
[1] 100
```

按索引取数据 first() 和 last()。

```
> x <- xts(1:100, Sys.Date()+1:100)
> head(x)
           [,1]
2013-11-19    1
2013-11-20    2
2013-11-21    3
2013-11-22    4
2013-11-23    5
2013-11-24    6

> first(x, 10)  # 取前10条数据
           [,1]
```

```
2013-11-19    1
2013-11-20    2
2013-11-21    3
2013-11-22    4
2013-11-23    5
2013-11-24    6
2013-11-25    7
2013-11-26    8
2013-11-27    9
2013-11-28   10

> first(x, '1 day')     # 取 1 天的数据
           [,1]
2013-11-19    1
> last(x, '1 weeks')  # 取最后 1 周的数据
           [,1]
2014-02-24   98
2014-02-25   99
2014-02-26  100
```

计算步长 lag() 和差分 diff()。

```
> x <- xts(1:5, Sys.Date()+1:5)
> lag(x)  # 以 1 为步长
           [,1]
2013-11-19   NA
2013-11-20    1
2013-11-21    2
2013-11-22    3
2013-11-23    4
> lag(x, k=-1, na.pad=FALSE)  # 以 -1 为步长，并去掉 NA 值
           [,1]
2013-11-19    2
2013-11-20    3
2013-11-21    4
2013-11-22    5

> diff(x)   # 1 阶差分
           [,1]
2013-11-19   NA
2013-11-20    1
2013-11-21    1
2013-11-22    1
2013-11-23    1
> diff(x, lag=2)  # 2 阶差分
           [,1]
2013-11-19   NA
```

```
2013-11-20   NA
2013-11-21    2
2013-11-22    2
2013-11-23    2
```

使用 isOrdered() 函数，检查向量是否排序好的。

```
> isOrdered(1:10, increasing=TRUE)
[1] TRUE
> isOrdered(1:10, increasing=FALSE)
[1] FALSE
> isOrdered(c(1,1:10), increasing=TRUE)
[1] FALSE
> isOrdered(c(1,1:10), increasing=TRUE, strictly=FALSE)
[1] TRUE
```

使用 make.index.unique() 函数，强制唯一索引。

```
> x <- xts(1:5, as.POSIXct("2011-01-21") + c(1,1,1,2,3)/1e3)
> x
                        [,1]
2011-01-21 00:00:00.000    1
2011-01-21 00:00:00.000    2
2011-01-21 00:00:00.000    3
2011-01-21 00:00:00.002    4
2011-01-21 00:00:00.003    5
> make.index.unique(x)  # 增加毫秒级精度，保证索引的唯一性
                           [,1]
2011-01-21 00:00:00.000999    1
2011-01-21 00:00:00.001000    2
2011-01-21 00:00:00.001001    3
2011-01-21 00:00:00.002000    4
2011-01-21 00:00:00.003000    5
```

查询 xts 对象时区。

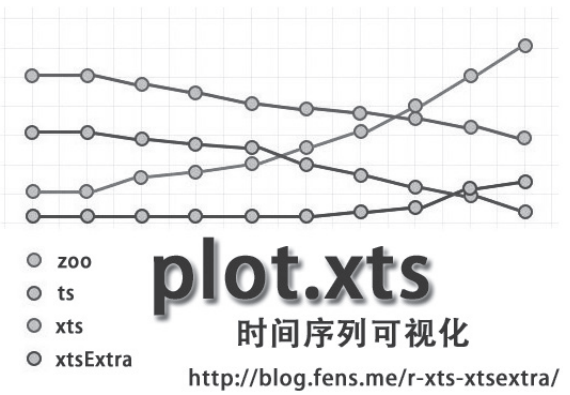
```
> x <- xts(1:10, Sys.Date()+1:10)
> indexTZ(x)  # 时区查询
[1] "UTC"
> tzone(x)
[1] "UTC"
> str(x)
An 'xts' object on 2013-11-19/2013-11-28 containing:
  Data: int [1:10, 1] 1 2 3 4 5 6 7 8 9 10
  Indexed by objects of class: [Date] TZ: UTC
  xts Attributes:
 NULL
```

xts 给了 zoo 类型时间序列更多的 API 支持，这样我们就有了更方便的工具，可以做各种时间序列的转换和变形了。

2.3　时间序列可视化 plot.xts

问题

如何把时间序列可视化?



引言

r-bloggers 的一篇博文：plot.xts is wonderful! (http://www.r-bloggers.com/plot-xts-is-wonderful/)，让我有动力继续发现 xts 的强大。xts 扩展了 zoo 的基础数据结构，并提供了更丰富的功能函数。xtsExtra 补充库从可视化的角度出发，提供了一个简单而效果非凡的作图函数 plot.xts。本节将用 plot.xts 来演示 xts 对象的时间序列可视化!

2.3.1　xtsExtra 介绍

xtsExtra 是 xts 包的功能补充包，该软件包在 Google Summer of Code 2012 发布，最终将合并到 xts 包，不过在笔者写这本书的时候还没有合并。xtsExtra 提供的主要功能就是 plot.xts() 函数。xts::plot.xts() 函数与 xtsExtra::plot.xts() 函数还是有差别的，下面我们就详细介绍其中的差别!

2.3.2　xtsExtra 安装

本节使用的系统环境是：

❑ Win7 64bit

❑ R: 3.0.1 x86_64-w64-mingw32/x64 b4bit

注 xtsExtra 同时支持 Windows 7 环境和 Linux 环境。

由于 xtsExtra 没有发布到 CRAN，我们要从 R-Forge 下载。

```
~ R  # 启动 R 程序
> install.packages("xtsExtra", repos="http://R-Forge.R-project.org")
# 从 R-Forge 下载 xtsExtra 包
> library(xtsExtra)  # 加载 xtsExtra 包
```

xtsExtra::plot.xts() 函数覆盖了 xts::plot.xts() 函数。

2.3.3 xtsExtra 包的使用

plot.xts() 函数的参数列表如下：

```
> names(formals(plot.xts))
 [1] "x"              "y"           "screens"      "layout.screens"  "..."
 [6] "yax.loc"        "auto.grid"   "major.ticks"  "minor.ticks"     "major.format"
[11] "bar.col.up"     "bar.col.dn"  "candle.col"   "xy.labels"       "xy.lines"
[16] "ylim"           "panel"       "auto.legend"  "legend.names"    "legend.loc"
[21] "legend.pars"    "events"      "blocks"       "nc"              "nr"
```

下面画一个简单的时间序列图，如图 2-10 所示。

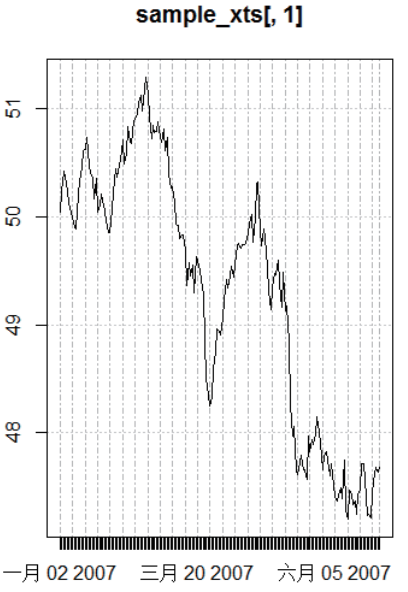


图 2-10 时间序列

```
> data(sample_matrix)
> sample_xts <- as.xts(sample_matrix)
> plot(sample_xts[,1])
> class(sample_xts[,1])
[1] "xts" "zoo"
```

从图 2-10 似乎看不出 xtsExtra::plot.xts() 函数与 xts::plot.xts() 函数的不同效果。接下来，我们画点稍微复杂的图形。

1. 画 K 线图

下面就来画 K 线图，默认红白配色，如图 2-11 所示。

```
> plot(sample_xts[1:30, ], type = "candles")
```

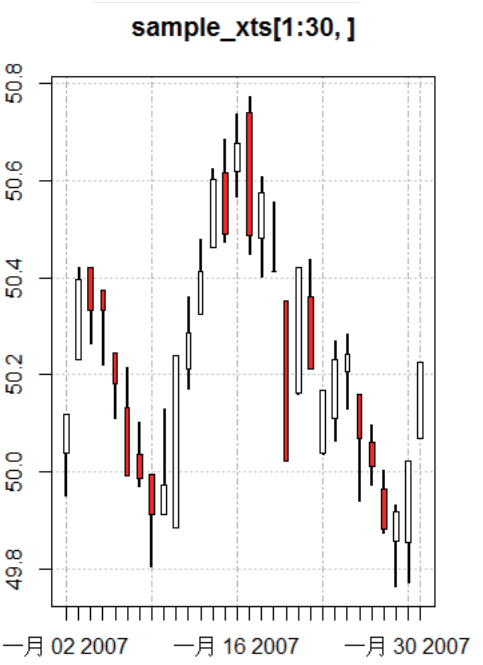


图 2-11　K 线图

画 K 线图：自定义颜色，如图 2-12 所示。

```
> plot(sample_xts[1:30, ], type = "candles", bar.col.up = "blue", bar.col.dn =
 "violet", candle.col = "green4")
```

2. 对 panel 配置

画基本面板，如图 2-13 所示。

```
> plot(sample_xts[,1:2])
```

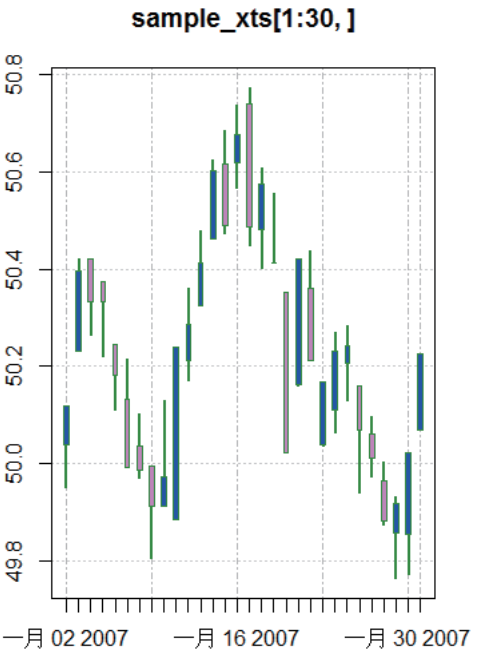


图 2-12 自定义颜色的 K 线图

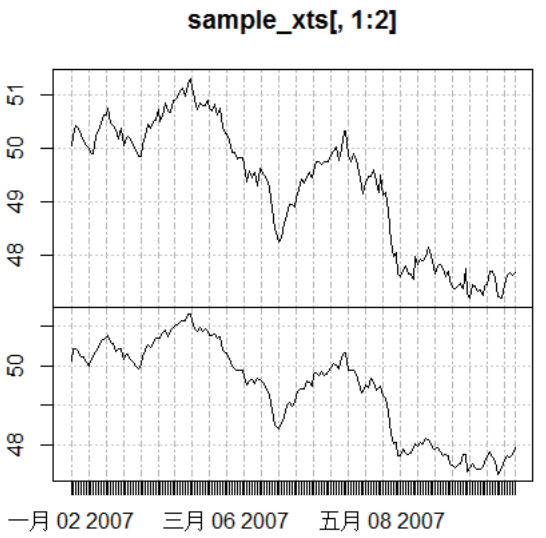


图 2-13 基本面板

画多行面板，如图 2-14 所示。

```
> plot(sample_xts[,rep(1:4, each = 3)])
```

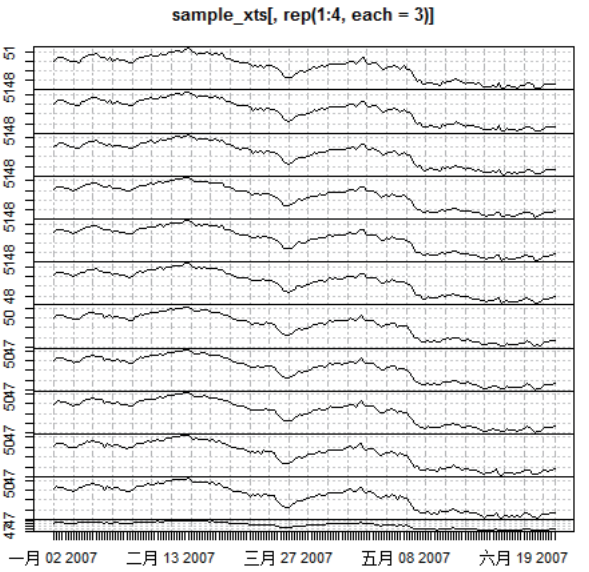


图 2-14　多行面板

画自由组合面板，如图 2-15 所示。

```
> plot(sample_xts[,1:4], layout.screens = matrix(c(1,1,1,1,2,3,4,4),ncol = 2,
 byrow = TRUE))
```

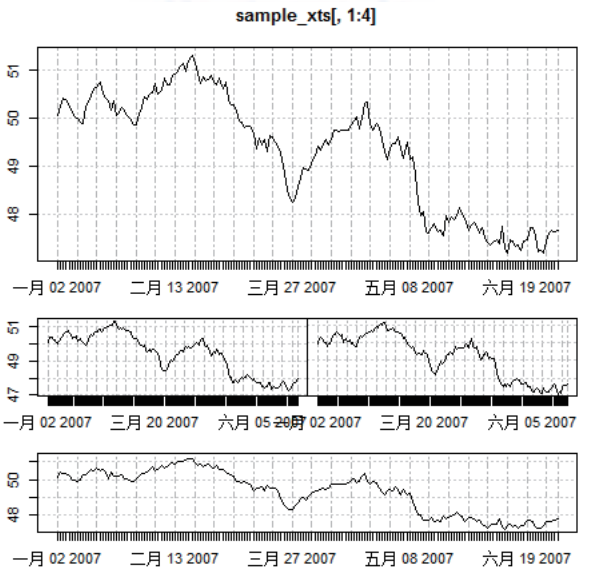


图 2-15　自由组合面板

通过画 K 线图和面板，就能发现 plot.xts() 函数提供很多种的画图参数设置，让我们画时间序列图可以有更丰富的可视化表示表现形式。

3. 对 screens 配置

画双屏幕显示，每屏幕 2 条线，如图 2-16 所示。

```
> plot(sample_xts, screens = 1:2)
```

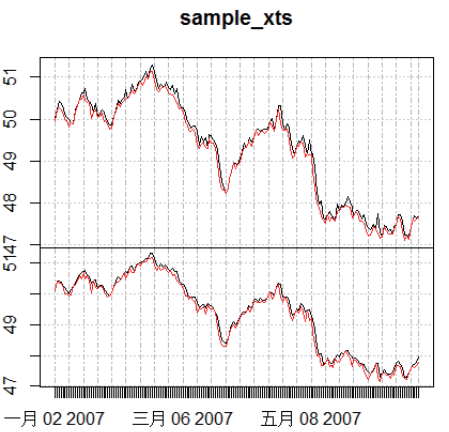


图 2-16 双屏显示

画双屏幕显示，指定曲线出现的屏幕和颜色，如图 2-17 所示。

```
> plot(sample_xts, screens = c(1,2,1,2), col = c(1,3,2,2))
```

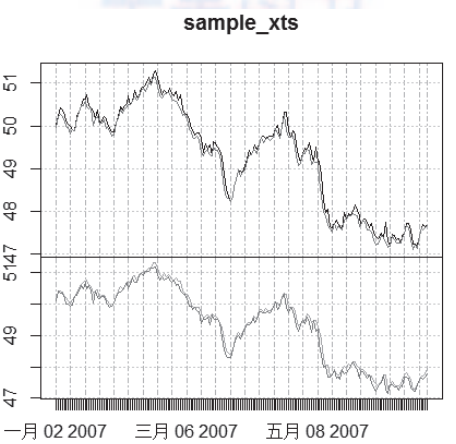


图 2-17 双屏显示且自定义颜色

画双屏幕显示，指定不同的坐标系，如图 2-18 所示。

```
> plot(10^sample_xts, screens = 1:2, log= c("","y"))
```

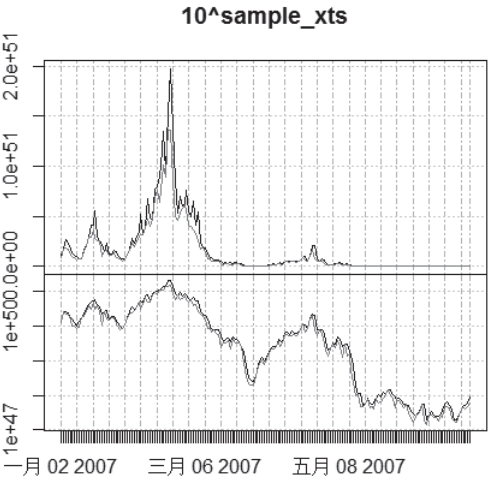


图 2-18　双屏显示，并设置不同坐标系

画双屏幕显示，指定不同的输出图形，如图 2-19 所示。

```
> plot(sample_xts[1:75,1:2] - 50.5, type = c("l","h"), lwd = c(1,2))
```

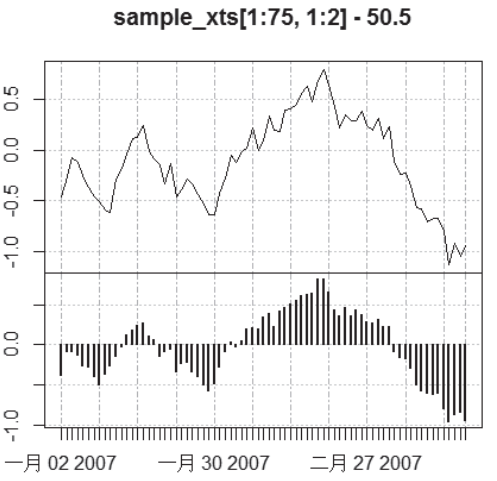


图 2-19　双屏显示且设置不同的输出图形

画多屏幕，并分组显示，如图 2-20 所示。

```
> plot(sample_xts[,c(1:4, 3:4)], layout = matrix(c(1,1,1,1,2,2,3,4,5,6), ncol = 2,
  byrow = TRUE), yax.loc = "left")
```

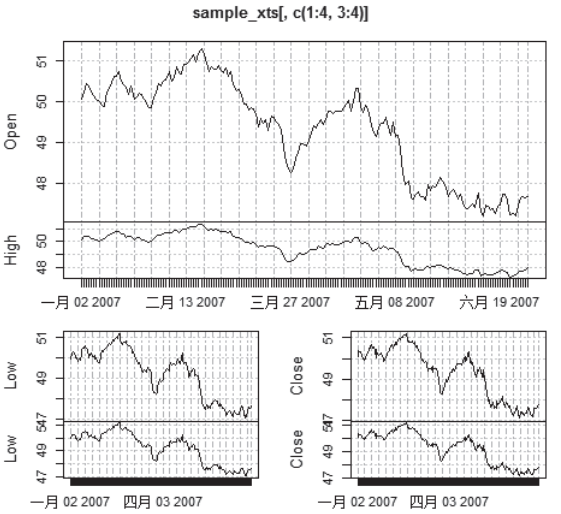


图 2-20 多屏显示，并多屏分组

4. 对 events 配置

画基本事件分割线，如图 2-21 所示。

```
> plot(sample_xts[,1], events = list(time = c("2007-03-15","2007-05-01"), label
  = "bad days"), blocks = list(start.time = c("2007-03-05", "2007-04-15"), end.
  time = c("2007-03-20","2007-05-30"), col = c("lightblue1", "lightgreen")))
```

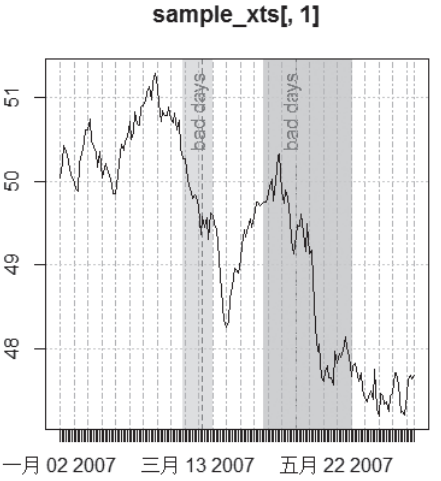


图 2-21 事件分割线

5. 双坐标的时间序列

画双坐标视图，如图 2-22 所示。

```
> plot(sample_xts[,1],sample_xts[,2])
```

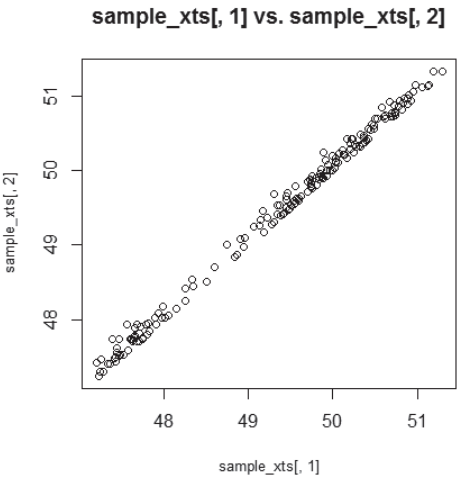


图 2-22　双坐标视图

画双坐标梯度视图，如图 2-23 所示。

```
> cr <- colorRampPalette(c("#00FF00","#FF0000"))
> plot(sample_xts[,1],sample_xts[,2], xy.labels = FALSE, xy.lines = TRUE, col =
  cr(NROW(sample_xts)), type = "l")
```

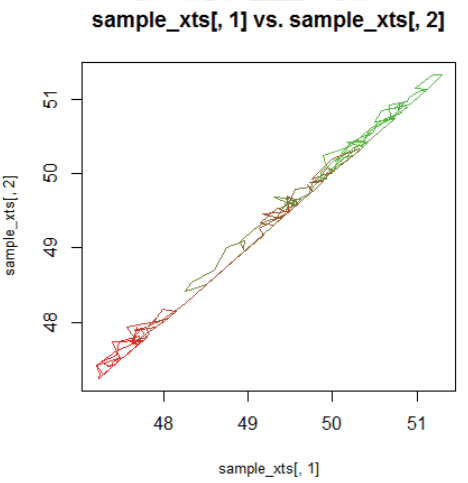


图 2-23　双坐标梯度视图

6. 对 xts 类型转换作图

以 ts 数据类型作图，如图 2-24 所示。

```
> tser <- ts(cumsum(rnorm(50, 0.05, 0.15)), start = 2007, frequency = 12)
> class(tser)
[1] "ts"
> plot(tser)
```

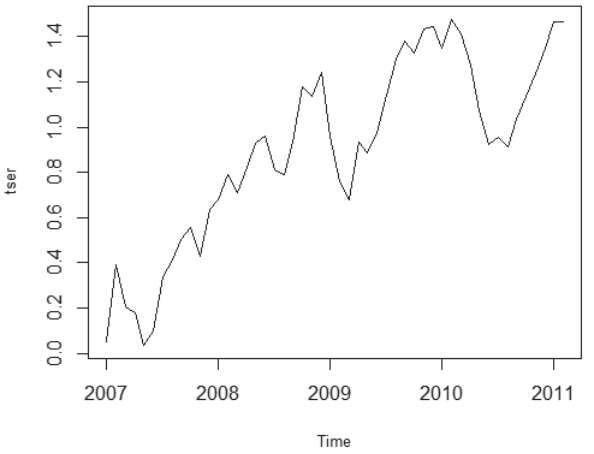


图 2-24 ts 数据类型作图

以 xts 类型作图，自动增加了背景坐标线，如图 2-25 所示。

```
> plot.xts(tser)
```

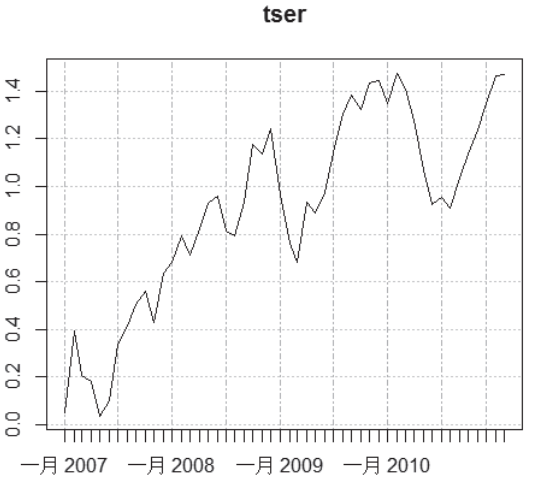


图 2-25 xts 类型作图

7. 用 barplot 作图

barplot() 函数作条形图，如图 2-26 所示。

```
> x <- xts(matrix(abs(rnorm(72)), ncol = 6), Sys.Date() + 1:12)
> colnames(x) <- LETTERS[1:6]
> barplot(x)
```

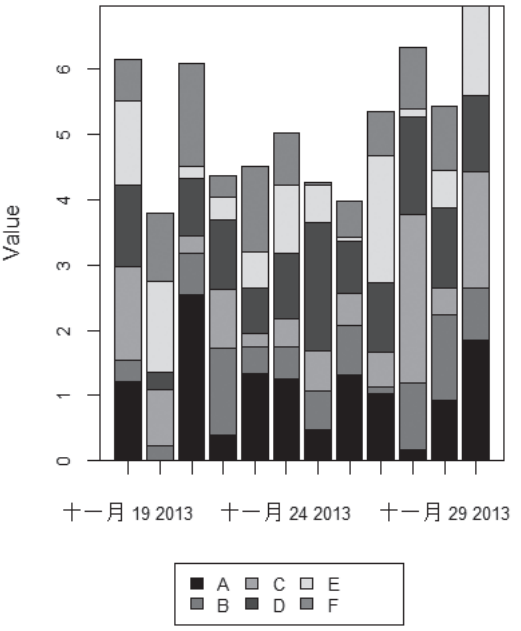
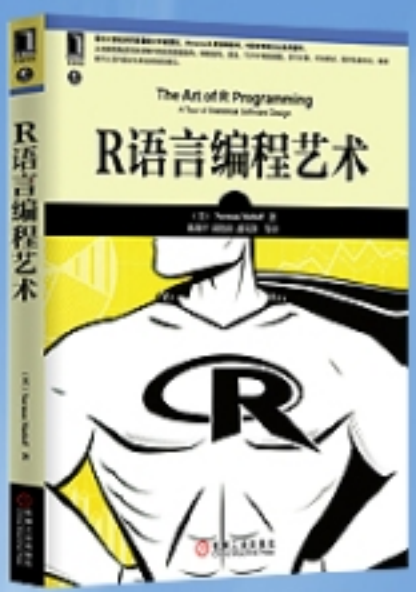
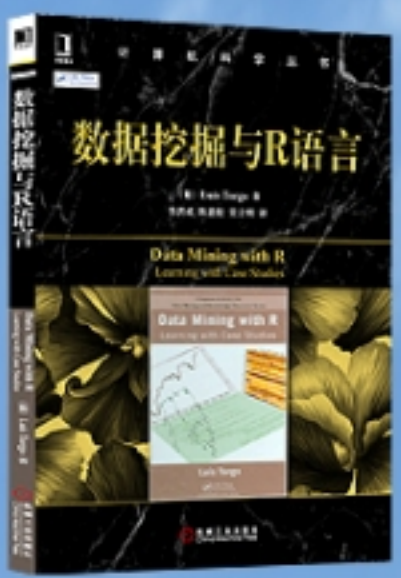


图 2-26　barplot 图

我们看到 xtsExtra::plot.xts() 函数提供了强大的作图功能，很容易做出包含丰富元素的时间序列！

相关图书推荐


R语言编程艺术
The Art of R Programming
R语言编程艺术


数据挖掘与R语言
Data Mining with R
Learning with Case Studies


机器学习
实用案例解析
O'REILLY


R语言
经典实例
O'REILLY


R的极客理想
工具篇


R语言与网站分析