

LEARNING PROCESSING

A Beginner's Guide to Programming
Images, Animation, and Interaction

Daniel Shiffman



学习Processing

一个初学者的图像编程、动画以及交互指导

摩根考夫曼系列计算机图形学

- Learning Processing Daniel Shiffman
Digital Modeling of Material Appearance Julie Dorsey, Holly Rushmeier, and François Sillion
Mobile 3D Graphics with OpenGL ES and M3G Kari Pulli, Tomi Aarnio, Ville Miettinen, Kimmo Roimela, and Jani Vaarala
Visualization in Medicine Bernhard Preim and Dirk Bartz
Geometric Algebra for Computer Science: An Object-oriented Approach to Geometry Leo Dorst, Daniel Fontijne, and Stephen Mann
Point-Based Graphics Markus Gross and Hanspeter Pfister, Editors
High Dynamic Range Imaging: Data Acquisition, Manipulation, and Display Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec
Complete Maya Programming Volume II: An In-depth Guide to 3D Fundamentals, Geometry, and Modeling David A. D. Gould
MEL Scripting for Maya Animators, Second Edition Mark R. Wilkins and Chris Kazmier
Advanced Graphics Programming Using OpenGL Tom McReynolds and David Blythe
Digital Geometry Geometric Methods for Digital Picture Analysis Reinhard Klette and Azriel Rosenfeld
Digital Video and HDTV Algorithms and Interfaces Charles Poynton
Real-Time Shader Programming Ron Fosner
Complete Maya Programming: An Extensive Guide to MEL and the C API David A. D. Gould
- Texturing & Modeling: A Procedural Approach, Third Edition David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley
Geometric Tools for Computer Graphics Philip Schneider and David H. Eberly
Understanding Virtual Reality: Interface, Application, and Design William B. Sherman and Alan R. Craig
Jim Blinn's Corner: Notation, Notation, Notation Jim Blinn
Level of Detail for 3D Graphics David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner
Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling Ron Goldman
Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation Thomas Strothotte and Stefan Schlechtweg
Curves and Surfaces for CAGD: A Practical Guide, Fifth Edition Gerald Farin
Subdivision Methods for Geometric Design: A Constructive Approach Joe Warren and Henrik Weimer
Computer Animation: Algorithms and Techniques Rick Parent
The Computer Animator's Technical Handbook Lynn Pocock and Judson Rosebush
Advanced RenderMan: Creating CGI for Motion Pictures Anthony A. Apodaca and Larry Gritz
Curves and Surfaces in Geometric Modeling: Theory and Algorithms Jean Gallier
- Andrew Glassner's Notebook: Recreational Computer Graphics Andrew S. Glassner
Warping and Morphing of Graphical Objects Jonas Gomes, Lucia Darsa, Bruno Costa, and Luiz Velho
Jim Blinn's Corner: Dirty Pixels Jim Blinn
Rendering with Radiance: The Art and Science of Lighting Visualization Greg Ward Larson and Rob Shakespeare
Introduction to Implicit Surfaces Edited by Jules Bloomenthal
Jim Blinn's Corner: A Trip Down the Graphics Pipeline Jim Blinn
Interactive Curves and Surfaces: A Multimedia Tutorial on CAGD Alyn Rockwood and Peter Chambers
Wavelets for Computer Graphics: Theory and Applications Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin
Principles of Digital Image Synthesis Andrew S. Glassner
Radiosity & Global Illumination François X. Sillion and Claude Puech
Knotty: A B-Spline Visualization Program Jonathan Yen
User Interface Management Systems: Models and Algorithms Dan R. Olsen, Jr.
Making Them Move: Mechanics, Control, and Animation of Articulated Figures Edited by Norman I. Badler, Brian A. Barsky, and David Zeltzer
Geometric and Solid Modeling: An Introduction Christoph M. Hoffmann
An Introduction to Splines for Use in Computer Graphics and Geometric Modeling Richard H. Bartels, John C. Beatty, and Brian A. Barsky

学习 Processing

一个初学者的图像编程、动画以及交互指导

Daniel Shiffman



ELSEVIER

阿姆斯特丹•波士顿•海德堡•伦敦
纽约•牛津•巴黎•圣地亚哥
旧金山•新加坡•悉尼•东京

摩根考夫曼出版商



Morgan Kaufmann Publishers is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA

Copyright © 2008, Elsevier Inc. All rights reserved.

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. All trademarks that appear or are otherwise referred to in this work belong to their respective owners. Neither Morgan Kaufmann Publishers nor the authors and other contributors of this work have any relationship or affiliation with such trademark owners nor do such trademark owners confirm, endorse, or approve the contents of this work. Readers, however, should contact the appropriate companies for more information regarding trademarks and any related registrations.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, scanning, or otherwise—without prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (44) 1865 843830, fax: (44) 1865 853333, E-mail: permissions@elsevier.com. You may also complete your request online via the Elsevier homepage (<http://www.elsevier.com>) by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."

Library of Congress Cataloging-in-Publication Data
Application submitted.

ISBN: 978-0-12-373602-4

For information on all Morgan Kaufmann publications, visit our
Web site at www.mkp.com or www.books.elsevier.com

Typeset by Charon Tec Ltd., A Macmillan Company (www.macmillansolutions.com).

Printed in the United States of America.

08 09 10 11 12 5 4 3 2 1

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER

BOOK AID
International

Sabre Foundation

目录

致谢	vii
介绍	ix
<hr/>	
第一课：初识	1
第1章：像素	3
第2章：Processing	17
第3章：互动	31
<hr/>	
第二课：你需要了解的一切	43
第4章：变量	45
第5章：条件	59
第6章：循环	81
<hr/>	
第三课：组织	99
第7章：函数	101
第8章：对象	121
<hr/>	
第四课：更多相同的	139
第9章：数组	141
<hr/>	
第五课：整合所有部份	163
第10章：算法	165
第11章：调试	191
第12章：库	195
<hr/>	
第六课：世界围绕着你	199
第13章：数学	201
第14章：平移和旋转（在3D!）	227
<hr/>	
第七课：在显微镜下的像素	253
第15章：图像	255
第16章：视频	275
<hr/>	
第八课：外面的世界	303
第17章：文本	305
第18章：数据导入	325

第19章：数据串

357

第九课：制造噪音

379

第20章：声音

381

第21章：导出

397

第十课：除了Processing之外

407

第22章：先进的面向对象编程

409

第23章：JAVA

423

附录：常见错误

439

索引

447

致谢

In the fall of 2001, I wandered into the Interactive Telecommunications Program in the Tisch School of the Arts at New York University having not written a line of code since some early 80's experiments in BASIC on an Apple II. There, in a first semester course entitled Introduction to Computational Media, I discovered programming. Without the inspiration and support of ITP, my home since 2001, this book would have never been written.

Red Burns, the department's chair and founder, has supported and encouraged me in my work for the last seven years. Dan O'Sullivan has been my teaching mentor and was the first to suggest that I try a course in Processing at ITP, giving me a reason to start putting together programming tutorials. Shawn Van Every sat next to me in the office throughout the majority of the writing of this book, providing helpful suggestions, code, and a great deal of moral support along the way. Tom Igoe's work with physical computing provided inspiration for this book, and he was particularly helpful as a resource while putting together examples on network and serial communication. And it was Clay Shirky who I can thank for one day stopping me in the hall to tell me I should write a book in the first place. Clay also provided a great deal of feedback on early drafts.

All of my fellow computational media teachers at ITP have provided helpful suggestions and feedback along the way: Danny Rozin (the inspiration behind Chapters 15 and 16), Amit Pitaru (who helped in particular with the chapter on sound), Nancy Lewis, James Tu, Mark Napier, Chris Kairalla, and Luke Dubois. ITP faculty members Marianne Petit, Nancy Hechinger, and Jean-Marc Gauthier have provided inspiration and support throughout the writing of this book. The rest of the faculty and staff at ITP have also made this possible: George Agudow, Edward Gordon, Midori Yasuda, Megan Demarest, Robert Ryan, John Duane, Marlon Evans, and Tony Tseng.

The students of ITP, too numerous to mention, have been an amazing source of feedback throughout this process, having used much of the material in this book in trial runs for various courses. I have stacks of pages with notes scrawled along the margins as well as a vast archive of e-mails with corrections, comments, and generous words of encouragement.

I am also indebted to the energetic and supportive community of Processing programmers and artists. I'd probably be out of a job if it weren't for Casey Reas and Benjamin Fry who created Processing. I've learned half of what I know simply from reading through the Processing source code; the elegant simplicity of the Processing language, web site, and IDE has made programming accessible and fun for all of my students. I've received advice, suggestions, and comments from many Processing programmers including Tom Carden, Marius Watz, Karsten Schmidt, Robert Hodgin, Ariel Malka, Burak Arikan, and Ira Greenberg. The following teachers were also helpful test driving early versions of the book in their courses: Hector Rodriguez, Keith Lam, Liubo Borissov, Rick Giles, Amit Pitaru, David Maccarella, Jeff Gray, and Toshitaka Amaoka.

Peter Kirn and Douglas Edric Stanley provided extraordinarily detailed comments and feedback during the technical review process and the book is a great deal better than it would have been without their efforts. Demetrie Tyler did a tremendous job working on the visual design of the cover and interior of this book, making me look much cooler than I am. And a thanks to David Hindman, who worked on helping

me organize the screenshots and diagrams.

I'd also like to thank everyone at Morgan Kaufmann/Elsevier who worked on producing the book: Gregory Chalson, Tiffany

Gasbarrini, Jeff Freeland, Danielle Monroe, Matthew Cater, Michele Cronin, Denise Penrose, and Mary James.

Finally, and most important, I'd like to thank my wife, Aliki Caloyerias, who graciously stayed awake whenever I needed to talk through the content of this book (and even when I felt the need to practice material for class) my parents, Doris and Bernard Schiffman; and my brother, Jonathan Schiffman, who all helped edit some of the very first pages of this book, even while on vacation.

介绍

这本书是什么？

这本书讲述了一件事。对于基础计算机编程的第一个步骤，编写你自己的代码，并在没有任何软件的束缚下创建你自己的多媒体互动。这本书不是给计算机科学家和工程师看的。这本书是专门为你的。

这本书是针对谁？

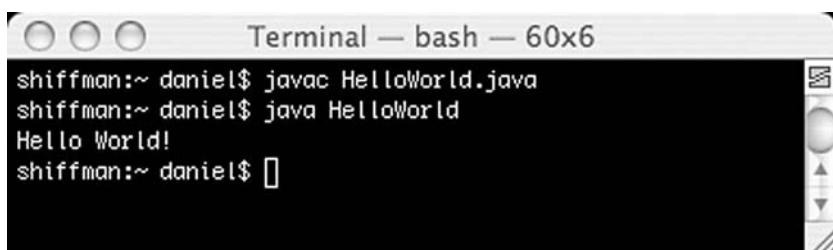
这本书是专门针对初学者。如果在你生命中还从来没有写过一行代码，这本书就是你要找的。没有假设，慢慢的一步一步从第1章到9章接触编程。除了计算机的基本操作你不需要任何专业背景 - 打开电脑、浏览网页、启动程序之类的就够了。

因为这本书使用的是 Processing (现在更多的是在 Processing 上操作) ，它用于学习或者工作在可视化领域的人特别的合适，如平面设计、绘画、雕塑、建筑、电影、视频、插画、网页这些。如果你深处在这些领域之中（至少有一个需要涉及到电脑），你可能必须需要一些特定软件，可能还不只一个软件，如 Photoshop, Illustrator, AutoCAD, Maya, After Effects 等。这本书就是希望你能从软件中释放出来，至少在一部分现在的局限上。你能做什么，你能做设计而不是运用别人的工具，是你自己写的工具。如果你感兴趣你，这本书就是你要找的。

如果你有一定编程经验，但是又有 Processing 的兴趣，这本书同样适用。你能从前面对的章节做一个快速的参考（有一个基础），并且你能在后面的章节有更多的应用优势。

什么是 Processing？

比方说你正在使用计算机科学，也许是 JAVA 编程语言。这里是在课堂上表示的第一个程序输出例子：



```
shiffman:~ daniel$ javac HelloWorld.java
shiffman:~ daniel$ java HelloWorld
Hello World!
shiffman:~ daniel$ []
```

传统上，程序员运用的基本知识是通过命令行输出：

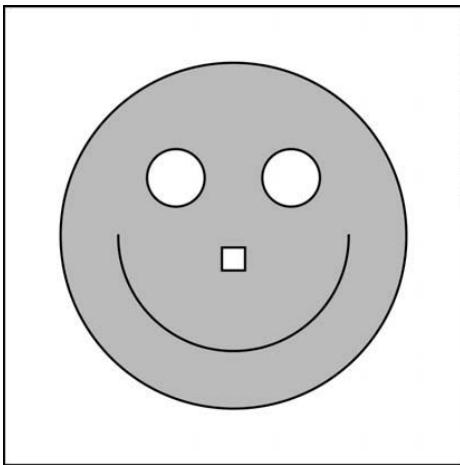
1. **TEXT 输入**→将你写的代码作为文本。
2. **TEXT 输出**→将你的代码生成文本输出在命令行上。
3. **文本相互作用**→用户可以与程序交互命令行上输入文字。

他输出了“Hello,World!”这个例子是一个老笑话了，一个程序员的惯例，你要学会在任何给定的语言中的第一个程序文本输出“Hello,World!”它首先出现在1974年贝尔实验室备忘录中，有Brian Kernighan编写的“C语言编程教程”。

Processing的重点放在了视觉响应的环境中，更有利于艺术家于设计师学习编程：

1. **TEXT 输入**→你写你的代码作为文本。
2. **视觉输出**→让你的代码在一个窗口中产生视觉效果。
3. **鼠标交互**→用户可以通过鼠标进行视觉交互（更多内容我们会从这本书看到！）。

Processing展现的“Hello, World!”可能是这个样子：



Hello, Shapes! (你好，形状！)

虽然看起来很有好，但是没有什么壮观的东西（这些初级编程都没有涉及到交互），但是总比“Hello, World!”强些，不过，我们学习到通过即时的视觉反馈是完全不同的概念。

Processing不是第一个运用这个范式的语言。在1967年，由 Daniel G. Bobrow、Wally Feurzeig以及 Seymour Papert 开发的Logo编程语言就运用过。在Logo语言中，程序员编写指示，就会在屏幕上缓慢的显示出来一些形状和设计。John Maeda's在1999年介绍了用数字计算去帮助视觉设计师以及艺术家提供一个简单易用的语法。

虽然这些语言是很好的简单以及创新，但是他们的能力是有限的。

Processing，一个直接继承Logo语言以及设计的语言在麻省理工学院媒体实验室2001年“美学与设计”研究小组中诞生。这是由Casey Reas和Benjamin Fry开发的一个开源语言。

Processing是一个开源的编程语言并且给予人们处理图像、动画、声音一个新的环境。它是为了让学生、艺术家、设计师、建筑师、研究人员和学习爱好者去学习、探索和开发。它创建了一个拥有电脑编程的基本原理的专业创作工具。Processing能作为代替那些那些专有软件固有的开发工具。

— www.processing.org

总而言之，Processing是一个很棒的东西。首先，它是免费的。他不会花费你一毛钱。其次，它是建立在JAVA编程语言的顶端（这一部分在最后一章由深入探讨），他是一个全功能的语言，没有数字设计软件和Logo语言的局限性。几乎没有什Processing不能做。最后，Processing是开源的。在大部分情况下，这些细节不是这本书的关键。然而，当你超越了初学者的阶段，这些哲学原理会是非常宝贵的。因为由开发人员、老师、艺术家在一起共享工作，献计献策、扩展开发能力。

你能在processing.org社区进行一个快速的学习和浏览。在哪里代码共享公开交流并且由一些艺术品被初学者和专家进行评论。虽然该网站包含了完整的参考目录，以及很多的例子，但是作为初学者，它不具有一步一步的教程。这本书的目的就是为了让你能有条不紊的学习编程的基本知识，能让你作出一步跳跃并能在社区的交流中作出贡献。

重要的是认识到没有专门针对Processing的成品书，这本书也不是一本Processing书，他的目的是为了教你编程。我们选择Processing作为我们的学习环境，但是重点在于计算的概念，这将带你进入数字生活，为您探索其他语言和环境提供帮助。

但是，我是不是可以学习_____？

你知道你想的。填写在空白出。你会听见在编程语言和环境中发生的大事。但是你的朋友不会停止谈论它怎么那么帮。他是怎么将一切变的那么简单。怎么让你一天的编程在5分钟内完成。以及能在Mac或者PC上甚至面包机上编程。

这里的东西，神奇的语言能让你的所有问题都解决掉。没有语言是完的，Processing也存在局限性和欠缺。但是它是一个很好的开始的地方。这本书教你编程的基本知识，这样你可以在你的一生中都能应用他们，包括Processing、Java、Actionscript、C、PHP或其他更多语言。

对于一些项目其他的语言以及可能更好。但是Processing真的能在很多东西上有不错的表现，尤其是媒体相关以及基于屏幕显示的作品。大家有一个常见的误解，Processing仅仅只能摆弄一些小东西；情况并非如此。人们（包括我自己）都在一年365天中使用Processing去做项目。它还被用于Web应用程序、在画廊和博物馆的艺术项目以及展品并在公共场所安装。最近我用Processing开发了一个实时图像视频幕墙系统(<http://www.mostpixelsever.com>)，它能显示120双脚的视频，它在纽约。

Processing不仅能做到很多很伟大的东西，而且没有比学习这些更好的了。它免费开源。又很简单，又是视觉方向。又很有趣。它是面向对象的（我们将在以后讨论），它能在Mac, PC以及Linux机器上运作。

所以我建议你，你不再担心它是什么，你应该集中精神学习Processing的基本内容。这方面的东西超出这本书将带你的任何东西。

写在这本书！

比方说，你是一个小说家。或者编剧。你想会花时间坐在电脑打字？或（喘气）打字机上打字？这是最有可能的。也许你会晚上躺在床上想着心中的想法，或者也许你喜欢坐在公园的长椅上，喂鸽子，想着心中的想法。一个深夜，在当地的酒吧，你会发现自己在一张餐巾纸上潦草地写着辉煌的情节转折。

嗯，写软件，编程，创建代码是没有什么不同。这是很容易忘记这一点，因为工作本身固有连接到计算机。但你必须找时间让你的思想做思考，思考逻辑，集思广益远离椅子，书桌，电脑。就个人而言，我做我最好的编程是在慢跑。

当然，实际计算机上打字是非常重要的。我的意思是，你最终不会改变生活本质的，工作奠定了基础。但你总以为是需要俯身研究液晶屏将是不够的。

写作这本书是一个正确的方向前进的一步，确保您通过代码将练习思想远离键盘。我又很多经验在这本书，纳入“填补空白”的做法。<http://www.learningprocessing.com>又这些空白的所有答案，所以你可以检查你是否正确地使用这些页面！当你有灵感时，作好记录，并把它写下来，可以把这本书当作一个笔记本。（当然，你也可以用你自己的笔记本）

我建议你话一般的时间原理电脑去读这本书再花一半的时间去单独写例子的代码。

我应该怎样读这本书？

最好是读完整本书。第1章，第2章，第3章，依此类推。你可以得到更多轻松一点的知识当第9章结束后，但在开始的时候是非常重要的。

这本书的目的时为了教你运用线性方式编程。更优化的编程就像一个参考。上半部分本书专用一个例子，并且建立它的特征，例例如一个步骤的时间（更多在这个例子上）。此外，计算机编程的根本元素都由一个特定的顺序来呈现，这里由一个精彩的互动电信项目（“ITP”），由一组纽约大学的学生完成的 (<http://itp.nyu.edu>)。

这本书的23个章节共分为10个课程。第1到第9章节介绍计算机图形，并且包含了计算机编程背后的基本原则。第10到12章节稍做停顿，去学习一些如何用增量的方法做更大项目的开发新的项目的资料。第13到23章扩展基本知识并从3D、视频、数据可视化中学习更高级的应用。

“课程”作为一个单位，标志着每一节课结束时我建议你暂停阅读下面内容，去将这节课程所学的应用在项目中。建议这样去做项目（但这都只是我的建议）。

这是一本教科书吗？

这本书可以作为入门级编程课程的教科书也可以作为自学手册。

我应该说明这本书的内容直接来自于ITP中的“计算媒体”。没有这些老师 (Dan O’ Sullivan, Danny Rozin, Chris Kairalla, Shawn Van Every, Nancy Lewis, Mark Napier, and James Tu)的帮助和数百名学生(我希望我能记住他们所有的名字)的帮助，我完全没想过会有这本书出现。

说句实在话，我包括了一些更多的素材能在一个学期之内讲完。除了23个章节以外，我大概还有18个详细素材在我的课程中（但是在这个书中都作为参考）。然而，无论是你在读书或者自学，你都需要在几个月中慢慢消化这些知识。当然，你可以很快速的读这本书，并且能够实际的编写代码开发项目，但这需要一个很大的时间量。由于快速所以你计划10天读完10个课程，这明显是不现实的！

下面是一个例子，它告诉你如何将这些材料在14周中学完。

第一周	第一课：1-3 章
第二周	第二课：4-6 章
第三周	第三课：7-8 章
第四周	第四课：第 9 章
第五周	第五课：10-11 章
第六周	期中考（同样继续第五课：第 12 章）

第七周	第六课：13-14 章
第八周	第七课：15-16 章
第九周	第八课：17-19 章
第十周	第九课：20-21 章
第十一周	第十课：22-23 章
第十二周	期末项目研讨会
第十三周	期末项目研讨会
第十四周	期末演讲

会需要考试吗？

一本书的能力只有那么多。关键在于实践、实践、再实践。把自己看作10岁并在上小提琴课。你的老师会告诉你每天练习。这似乎完全合情合理。每天做这本书中的练习，如果你可以的话。

有时候你在看的时候很难动手做出自己的想法。去做这些练习，你就不会这样想了。不过如果你有好的想法并且想要开发，你应该自由调整练习内容，以适应你在做的东西。

大量的练习知识一个几分钟的小小的演习，。有时候可能有点难，会长达一个多小时。在工作时，如果有一个很好的项目，它可能需要更长时间，几小时，一天或者一个星期。正如我刚才说的，需要一个良好的“课程”路线。我建议在每节课之间，你暂停阅读，在Processing上进行实践操作。

所有的测试题答案你都能在这本书的网站上找到。说到这...

你有网站吗？

这本书的网站是：<http://www.learningprocessing.com>

在网站里面你能发现：

- 在这本书的所有代码可下载的版本
- 书中的例子的在线版本（可以放在网上）
- 这本书中的任何错误的更正
- 在书中没有的的其他技巧和教程
- 问题和意见

由于很多在这本书中的例子使用的是彩色和动态的，所以这里只能给你提供黑白的静态截图，不会有完整的显示。你可以参考网站，并且可以在浏览器中运行那些例子或者将他们下载到本地运行。

这本书的网站没有很全的资料。Processing的官方网站：<http://www.processing.org>，你能发现许多许多的例子以及一个热闹的论坛可以讨论。

这本书的网站没有很全的资料。Processing 的官方网站：<http://www.processing.org>，你能发现许多许多的例子以及一个热闹的论坛可以讨论。

第一课

初识

- 1.像素
- 2.Processing
- 3.互动



1. 像素

“千里之行始于足下。”

—老子

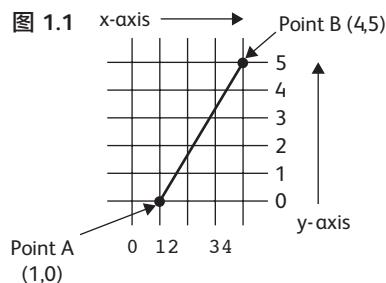
本章内容：

- 指定像素坐标
- 基本形状：点，线，矩形，椭圆
- 颜色：灰度，RGB
- 颜色的透明度

请注意在本章我们并没有做任何编程！我们只是试水，并且是结合文本的命令，了解“代码”！

1.1 Graph Paper

这本书将教你如何在计算机下进行媒体编程，并且会利用 Processing 的开发环境 (<http://www.processing.org>) 为基础建立所有的讨论以及例子。但是将这个变成兴趣之前，我们首先要引导我们自身，拿出一张纸，画一条线。两点之间是最短的距离的老式线段，我们就从这开始，在平面上连接2点。



```
line(1,0,4,5);
```

恭喜你，你已经写了第一行代码！我们将运用以上精确的公式，虽然我们现在对它们了解不多。我们为线段提供了一个命令“line”（我们将它称为“功能”）。此外我们指定了那个跟线段应该有的参数从 A (0,1) 到 B (4,5)。如果你将线段的代码是一句话，那么在这句话中函数就是动词，参数就是名词。代码结束后使用分号而不是句号。

图 1.2

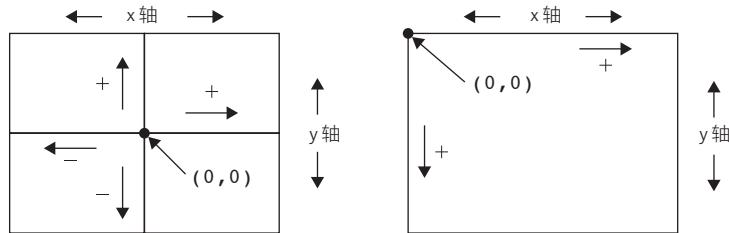
画一条线段从 0,1 到 4,5



这里关键是要认识到，在计算机屏幕上不像一张纸。每一个屏幕像素都是2个数值，即“x”值（水平）和“y”值（垂直），这2个数值决定点的位置。我们之后要做的就是指定各种形状以及颜色出现在这些像素坐标上。

然而，这里有一个例外。我们普通的坐标系（“笛卡尔坐标系”）是将(0,0)放置画面中间，y轴向上x轴向右，并且都有负方向值。而在计算机窗口的像素坐标系中却是相反的走向，并且(0,0)在屏幕的左上角，沿着右边和下边方向走，并且没有负方向值。参考图1.3。

图 1.3



练习1-1：参考我们写线段的代码“`line(1,0,4,5);`”你能猜出来怎么绘制一个矩形？一个圆？或者一个三角？试一试，写出中文的说明，然后将它转变为代码。



中文

代码

中文

代码

中文

代码

1.2 简单图形

在这本书中绝大多数的编程示例将用视觉方式呈现。你可能会学会怎么用 Processing 处理互动游戏的开发，艺术作品的开发，以及开机动画的设计。但是这每一个视觉项目都会涉及到怎么去设置像素。上手工作实际项目最简单的办法就是从简单的形状开始接触学习。这不像我们在小学画画，我们在这里使用代码，而不是蜡笔。

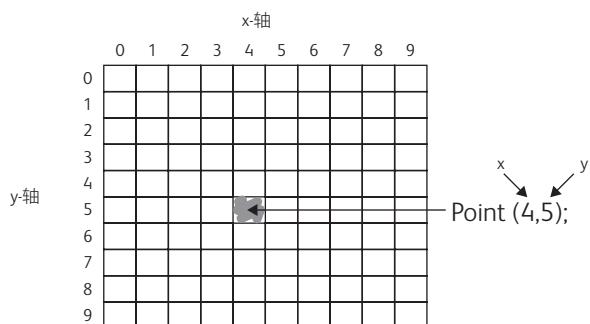
让我们开始了四个基本形状的学习，如图1.4所示。

图 1.4



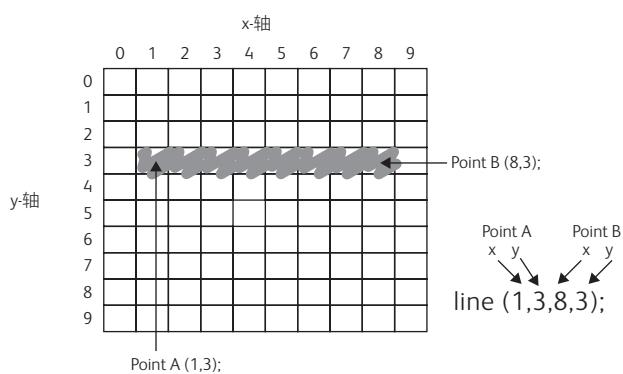
对于这些形状，我们肯定会问它们需要什么样的代码才能出现在指定的位置、尺寸、形状（以及之后会学到的颜色），并且怎么让 Processing 处理这些代码。看到下面的图（图1.5到 图1.11），假设一个有一个窗口，它的宽度为10px，高度为10px。这邮电不是很现实，因为当我们真正开始编程的时候我们的窗口可能很大（10px X 10px 只有几毫米屏幕大小）。由于我们在这里是演示，所以用较小的窗口就足够。这样也为了让我们更方便的说明每一行代码。

图 1.5



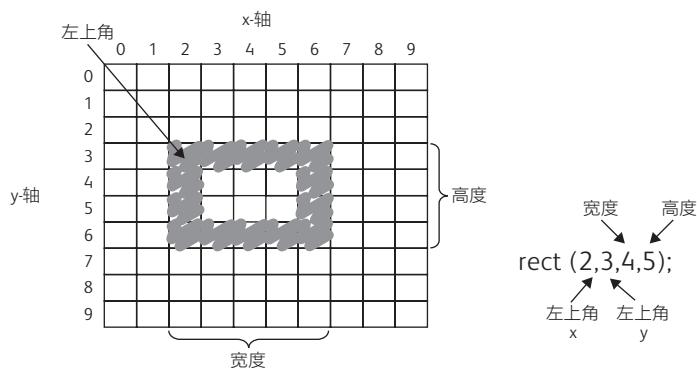
点是最简单的形状，对我们来说是一个很好的开端。要绘制一个点我们只需要知道它的x和y轴的坐标，如图1.5所示。如果要绘制一条线段其实也不难，线段只需要知道2个点的坐标即可，如图1.6所示。

图 1.6



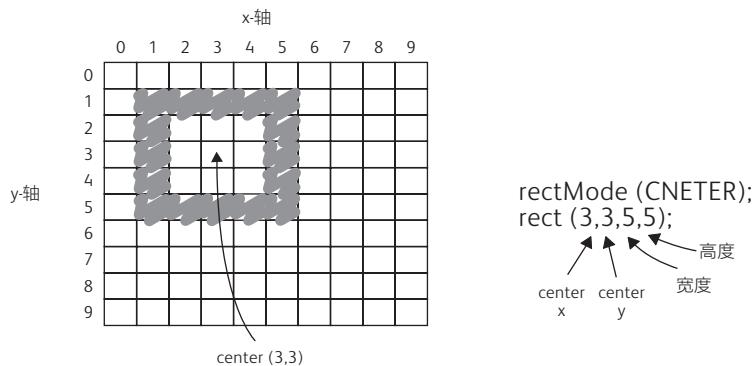
一旦到了我们绘制矩形，事情就开始变得有些复杂了。在 Processing 中，矩形有一个特定的坐标轴会指定在左上角，同样也需要有宽度和高度。

图 1.7



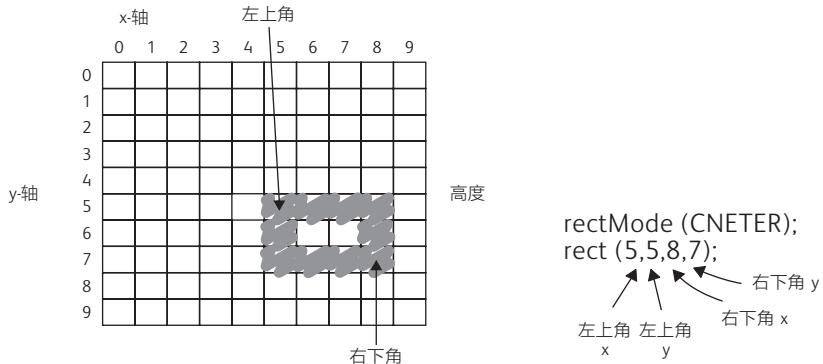
并且，我们还有另一种方式绘制矩形，如图1.8所示。我们将矩形的中心设置为其坐标，如果我们要使用这种方式绘制矩形必须要声明我们要使用“CENTER”（居中）模式，在书写时请注意区分大小写。再者，我们绘制的时候默认模式是“CORNER”（左上角），就是我们一开始的那样，如图1.7所示。

图 1.8



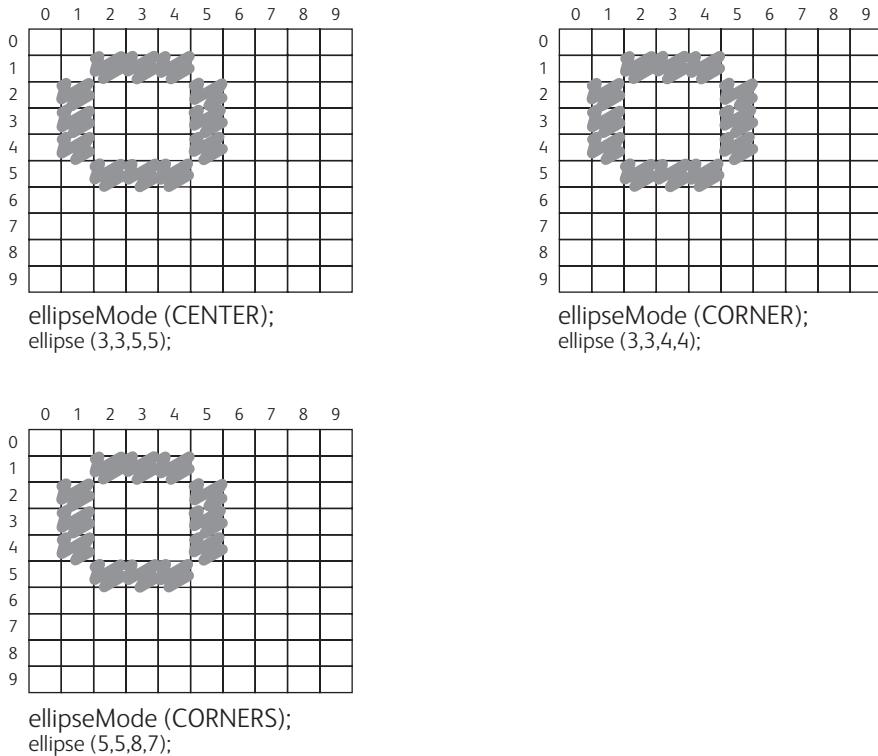
最后，我们还可以有两点绘制方式绘制矩形（左上角和右下角）。这里的模式是“CORNERS”，如图1.9所示。

图 1.9



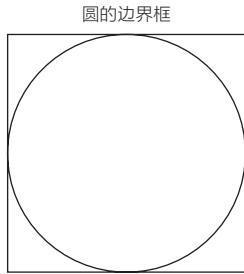
一旦我们对绘制矩形的概念熟知，那么我们绘制椭圆就成了我们新的单元。实际上它们是相同的，和rect()不同的是椭圆有一个矩形边界框（如图1.11所示）。ellipse()默认的绘制模式是“CENTER”（居中），而不是rect()中的“CORNER”（角落），如图1.10所示。

图 1.10



最重要的是你能看到在图1.10中，显示的圆不是圆形。Processing 有一个内置方式，用于选择那些可以创作圆形的像素点。放大之后，我们会看到一堆方形马塞克，但是缩小在电脑屏幕上的时候我们能看到的是一个很圆滑的圆。之后我们能够看到 Processing 神奇的像素着色方式（实际上，我们已经可应想到是怎样运行的了），但现在，请精神集中在绘制圆上面。

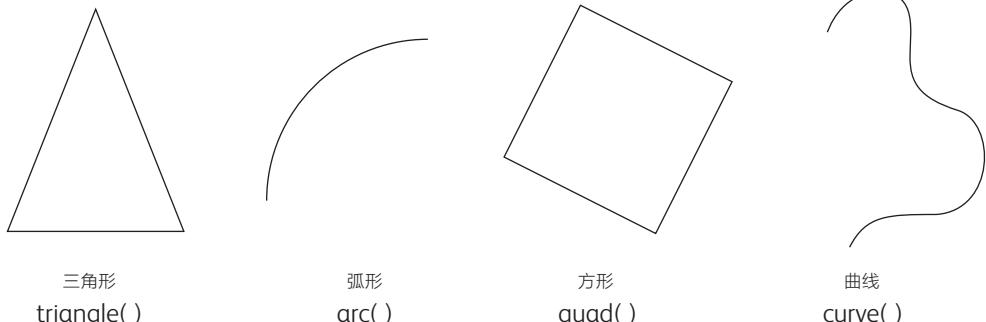
1.11



圆的边界框

当然在 Processing 的函数库中可以处理的不仅仅只有点、线、圆以及矩形这写简单的形状。在第二章中我们会学习到 Processing 提供的绘制功能的完整函数列表以及它们所需要参数。所以，现在你们可以试一试去想想它们（图 1.12）都需要怎样的参数，把它当作一个小练习。

圖 1.12



三角形
triangle()

弧形
arc()

方形
quad()

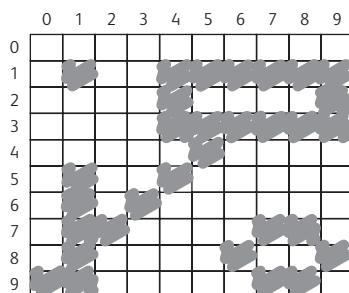
曲线
curve()



练习1-2：使用白纸，绘制下列代码。

```
line(0,0,9,6);
point(0,2);
point(0,4);
rectMode(CORNER);
rect(5,0,4,3);
ellipseMode(CENTER);
ellipse(3,7,4,4);
```

练习1-3：通过下面绘制出来的图像，用代码形式表现出来。



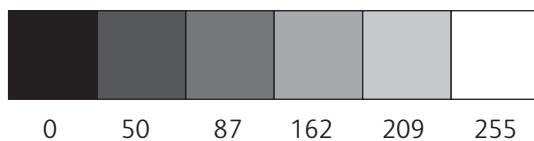
注：这里有不止一个的正确答案！

1.3 灰度色彩

正如我们了解到，第1.2节中的我们在屏幕上显示的基本单位是像素。当你指示电脑在一个规定大小和一个特定位置绘制一个形状时会发现一个基本要素是缺少的，它就是颜色。

在编程中我们对精确度要求很高，所以你不会说“嘿，你能给那个圆上一个蓝绿色吗”之类的话。因此，我们对于色彩值有很明确的数字定义。当我们举一个最简单的例子：黑色白色或者灰色。在灰色方面我们有一下几个数字：0表示黑色，255表示白色，在这之间每一个其他数字如50, 87, 162, 209等表示的是从黑到白的灰色范围。如图1.13所示。

图 1.13



对你来说0 - 255这些东西到底是什么含义？

图形填充的颜色数值会存储在计算机中。这些存储数据仅仅是一个0和1的长串字符（一大堆的开启或关闭开关）这些开关每一个是一位（1bit），8个在一起为一字节（1byte）。假设我们有8位（1字节）—我们有多少种可能去配置这些开关？答案是256种可能，或者说是在0到255之间的可能。所以对于灰度颜色我们会使用8位颜色值，对于全彩颜色我们会使用24位颜色值（红色，绿色以及蓝色部分各为8位颜色值；如图1.4所示）。

了解了这些颜色值是如何工作的，我们现在可以很轻松的给1.2节中的形状设置一个灰度色彩值。在Processing中，每一个形状都有一个stroke()以及fill()或者2者都同时存在。stroke()表示这个形状的轮廓线，fill()表示这个形状的内部状态。线和点只拥有stroke()属性，原因很明显。

如果在绘制图形是我们忘记定义颜色，Processing会自动为其stroke()填充黑色(0),fill()填充白色(255)。请注意我们现在会使用更多的实际数字填充像素，假设一个更大的窗口，尺寸为200x200px。如图1.14所示。

```
rect(50,40,75,100);
```

图 1.14



请在绘制形状之前添加stroke()以及fill()函数，这样我们可以设置形状的颜色。这就很像你在告诉你的一个朋友使用什么颜色的笔去在白纸上绘制图形。所以你会在画图形之前告诉他这些基本信息，而不是等画完之后。

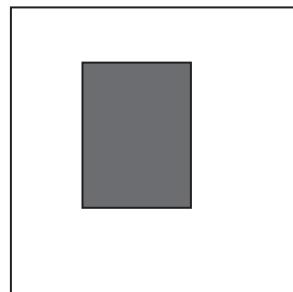
这里同样也包含background()这个函数，可以为窗口设置背景颜色。

例 1-1: stroke 和 fill

```
background(255);
stroke(0);
fill(150);
rect(50,50,75,100);
```

stroke()和fill()也能够消除变成noStroke()和noFill()函数。对于不要边框我们直觉上可能会说“stroke(0)”然而请记住重要的一点：0不是表示没有，而是表示颜色是黑色。另外请记住他们2个不能同时消除，如果noStroke()和noFill()那就什么都不会显示了！

图 1.15



例 1-2: noFill()

```
background(255);
stroke(0);
noFill();
ellipse(60,60,100,100);
```

如果我们再同一时间绘制2个形状，Processing讲始终选用最接近stroke()和fill()函数的形状，阅读代码顺序是从上自下的。如图1.17所示。

图 1.16

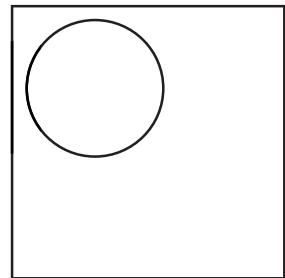
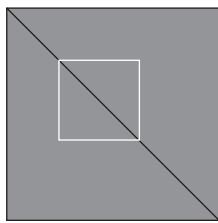
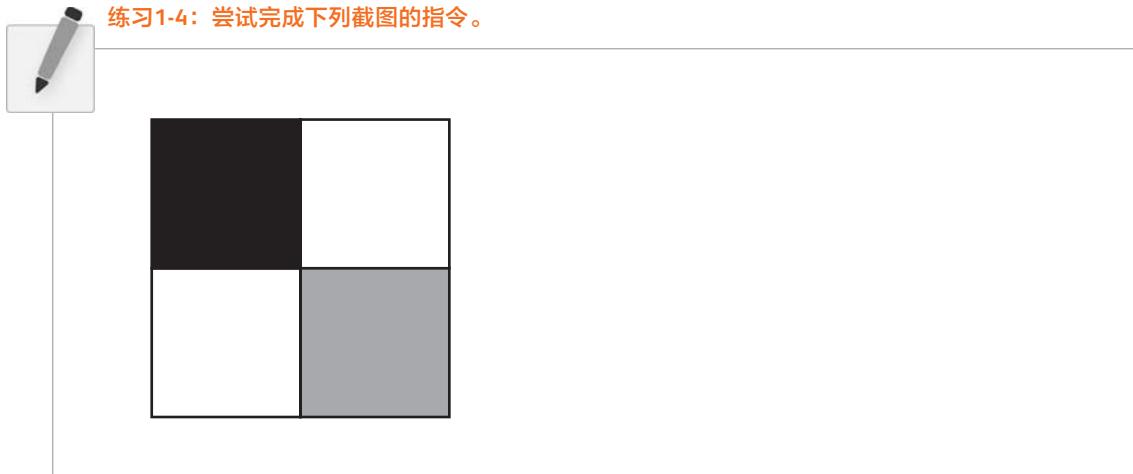


图 1.17



```
background(150);
stroke(0);
line(0,0,100,100);
stroke(255);
noFill();
rect(25,25,50,50);
```

练习1-4：尝试完成下列截图的指令。



1.3 RGB色彩

请随时看看之前学的，能够帮助我们学习像素位置以及尺寸的更多内容。现在让我们来研究学习数字色彩的基本知识，让我们开始寻找我们另一种童年记忆。还记得手指画吗？三个主要的颜色进行混合，可以产生任何颜色。叠加多个颜色，颜色越弄它的颜色就越深。

数字色彩同样也是由三原色构成，但是他的工作原理和绘画不同。首先，这些原色是不同的：红，绿，蓝（即RGB色彩）。颜色是在屏幕上显示，而不是画画，所以混合的规则也有很多不同。

■ Red +	■ green =	■ yellow
■ Red +	■ blue =	■ purple
■ Green +	■ blue =	■ cyan (blue-green)
■ Red +	■ green + ■ blue =	□ white
No colors =		black

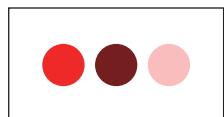
假设这些颜色都很明亮，当然你也有一系列颜色，所以一些红色加上一些绿色再加上一些蓝色就是灰色了，一点点红加上一点点蓝色等于深紫色。.

虽然在编程中使用RGB颜色可能需要一些时间去适应，但慢慢地，它的使用会成为我们的习惯，就像用你的手指作画一样。当然你不能说“混合一些红和一点点蓝”，你必须提供确切的数值。从0到255的数值，他们被列成R,G,B的顺序，你可以使用RGB颜色进行各种混搭，但接卸来我们将介绍一些常见的颜色代码。

例 1-3：RGB color

图 1.18

```
background(255);
noStroke();
```



```
fill(255,0,0);
ellipse(20,20,16,16);
```

红色

```
fill(127,0,0);
ellipse(40,20,16,16);
```

深红色

```
fill(255,200,200);
ellipse(60,20,16,16);
```

粉色

Processing同样也有颜色选择工具帮助选择颜色。通过这个工具（菜单栏中）→COLOR SELECTOR。如图1.19所示。

图 1.19



练习1-5：完成下列编程。猜测怎么填写这些RGB值（在下一章你可以用布尔值查看你的结果）。你同样也能使用color selector，如图1.19所示。



```

fill(_____,_____,_____);           蓝色
ellipse(20,40,16,16);

fill(_____,_____,_____);           深蓝色
ellipse(40,40,16,16);

fill(_____,_____,_____);           黄色
ellipse(60,40,16,16);

```

练习1-6：以下几行代码所显示的分别是什么颜色？



```

fill(0,100,0);      _____
fill(100);          _____
stroke(0,0,200);    _____
stroke(225);        _____
stroke(255,255,0);  _____
stroke(0,255,255);  _____
stroke(200,50,50);  _____

```

1.5 颜色透明度

在红色、绿色和蓝色之外，我们还有一个额外的选项，我们称之为色彩“alpha”。Alpha表示透明度以及当你想绘制的元素想透过底层的时候。

这里我们需要意识到，像素不会是字面上我们理解的透明，这是通过颜色混合产生的视觉错觉。在运行过程中，Processing会使用颜色数值以及一个0%到100%的比例建立混合光学感知。（如果你有对这些有兴趣，“rose-colored”眼镜是一个很好的开始。）

Alpha值同样是范围从0到255,0表示完全透明，255表示完全不透明。例1-4中的代码显示了图1.20。

例 1-4：Alpha 透明度

```
background(0);
noStroke();
```

```
fill(0,0,255);
rect(0,0,100,200);
```

没有第四个参数表示100%不透明

```
fill(255,0,0,255);
rect(0,0,200,40);
```

100%不透明

```
fill(255,0,0,191);
rect(0,50,200,40);
```

75%不透明

```
fill(255,0,0,127);
rect(0,100,200,40);
```

50%不透明

```
fill(255,0,0,63);
rect(0,150,200,40);
```

25%不透明

图 1.20



1.6 自定义颜色范围

从0到255范围的RGB色彩不是你能在Processing中能够唯一使用的范围。在计算机内存中，颜色经常是一系列24位（或者带有透明度32位）代码。但是Processing能够让我们把我们喜欢的颜色转换成计算机可以理解的代码。例如你可能更想要0到100范围的颜色（就像一个百分比）。你可以通过自定义一个色彩模式colorMode()。

colorMode(RGB,100); 用colorMode(), 你可以设置你自己的色彩范围。

上面的函数在说：“OK，我们想用红色，绿色，蓝色，并且他们的范围是从 0 到 100。”

虽然如上简单的设置很方便，但你也可以给予每个原色有不同的范围：

```
colorMode(RGB,100,500,10,255);
```

这时候我们就表示的是“红色的范围是从 0 到 100，绿色的范围是从 0 到 500,蓝色的范围是从 0 到 10 ,透明度是从 0 到 255”。

最后，对于模式模式的指定，一般情况下是使用 RGB 模式，也可以使用 HSB 模式。HSB 色彩模式详细如下：

- **Hue (色调)**—默认情况下色彩范围从 0 到 360（你可以理解为 360 度范围）。
- **Saturation (饱和度)**—默认情况下饱和度的范围从 0 到 100。
- **Brightness (亮度)**—默认情况下色彩的亮度范围从 0 到 100。

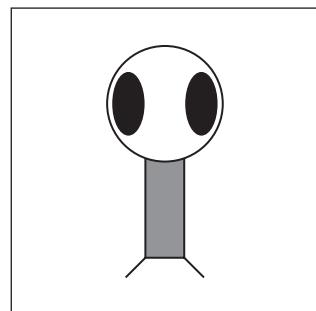
练习1-7：请用简单的形状和颜色设计出一个有趣的动物。先用手在纸上利用点、线、矩形、圆形划出动物，然后尝试用代码写出来，利用我们本章中介绍的命令：`point()`, `lines()`, `rect()`, `ellipse()`, `stroke()`, `fill()`。在下一章，你将学会如何测试你所写的 Processing 代码。



例 1-5：展示了我制作的 Zoog，如图 1.21 所示。

```
ellipseMode(CENTER);
rectMode(CENTER);
stroke(0);
fill(150);
rect(100,100,20,100);
fill(255); ellipse(100,70,60,60);
fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
stroke(0);
line(90,150,80,160);
line(110,150,120,160);
```

图 1.21



这个例子中的形象是我从 Processing 创造的，它叫 Zoog。从 1 到 9 章，我们会将我们所学的编程知识应用到它身上，你会看见它在长大。首先，我们将学习显示 Zoog，然后做出一个互动的 Zoog 以及一个动画的 Zoog，最后在画面中重复很多不同的 Zoog。

我建议你设计出属于自己的 Zoog（注意我们对你设计的动物或形状没有任何限制），并且和我一样在前 9 章的例子中都重复运用它。你只需要在每个例子中修改很少的一部分。这样能够有助于巩固计算机编程的变量，条件语句，循环语句，函数，对象，数组等基本要素，当你的 Zoog 逐渐熟练时，你可以放手去做这本书第 10 章开始的其他的任何主题。

2. Processing

“在未来的计算机可能不会超过1.5吨重。”

—《大众机械师》,1949

本章内容：

- 下载和安装 Processing
- 菜单选项
- 创建一个 Processing 草图文件
- 书写代码
- 查看错误
- Processing 参考
- 播放按钮
- 你的第一个草图
- 将你的草图发布到网上

1.3 Processing

现在我们知道了一些原始形状以及 RGB 色彩的知识，我们已经准备好在现实编程中应用这些知识。令人高兴的时对我们来说我们即将在 Processing 的环境中使编写代码，它是由 Casey Reas 2001 年在麻省理工学院媒体实验室开发的免费的，开源软件。（请查看这本书的介绍，并了解 Processing 的历史）

Processing 核心函数会在屏幕上显示出来，并且提供即时视觉反馈，并且让我们知道代码在做什么。因为它的编程语言和其他编程语言（特别是 Java）采用的是相同的原则和结构，你在 Processing 中学到的是真正的编程。它不是模拟的编程语言，它拥有所有的基本语言和概念。

读完这本书并且学习到编程，你可能会在你的专业或工作上继续使用 Processing 作为你的一个工具。你可能还会运用这里所学的只是并且把它应用到实践中。实际上，你可能还会发现编程不是你的口味，然而学习基础只是将有助于你更好的与其他的设计师和程序员合作项目。

看起不断来强调 Processing 有点小题大做。毕竟这本书的重点是在计算机图形和设计的背景下学习计算机编程的基础知识。但是花时间去思考背后的原因这很重要。因为现在你要考试成自己为计算机程序员，并且进行编程，以完成某个项目。

2.2 怎么获得Processing？

在大多数情况下，这本书都是人为你有基本的电脑操作基础。当然Processing是可以免费下载的。请访问 <http://www.processing.org> 进行下载。如果你是Windos用户你会看到两个选项：“Windos（标准版）”和“Windos（专家版）”。既然你正在读这本书，就相当于你是一个初学者，在这种情况下你可能会下载标准版本。专家版本已是为那些已经安装好Java的同志们准备的。对于Mac OS X，这里只有一个下载选项。这里同样也有一个Linux系统的下载选项。

Processing软件是一个经过压缩包压缩过。所以请选择一个合适的目录来存储这些应用程序（通常在Windos下是“c:\Program Files\”，在Mac下是“Applications”）解压之后，找到“Processing”的执行文件，并运行它。



练习2-1：下载并且安装Processing。

2.3 Processing 的应用

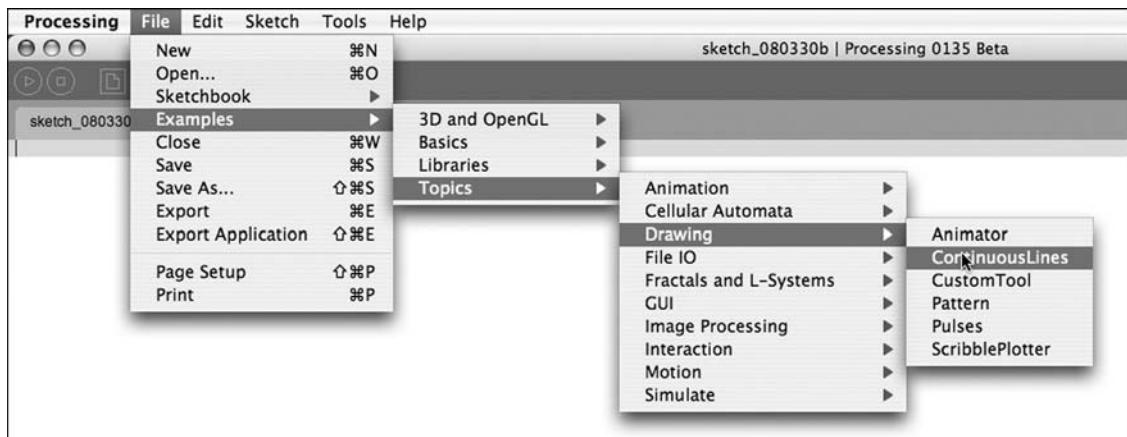
Processing 开发环境是一个简化的计算机代码环境，它只需要使用简单的文本编辑软件即可（如 Textedit 或者 Notepad）结合一个媒体播放器。每一个草图（ Processing 的编程文件被称为“草图”）都有一个文件名称，你能够把你所写的代码放入到草图中，并且存储、打开以及运行。如图 2.1 所示。

图 2.1



为了确保软件正常运行,请尝试运行Processing自带的例子。File → Examples → (打开一个例子: Topics → Drawing → ContinuousLines)。如图2.2所示。

图 2.2



一旦你打开了示例,请点击“run”按钮,如图2.3所示。如果有一个新的窗口弹出并且运行示例,说明你的软件没问题。如果没有发生什么,请访问在线答疑“Processing won't start!”了解可能的解决方案。你可以直接点击:<http://www.processing.org/faq/bugs.html#wontstart>

图 2.3



练习2-2: 从 Processing 打开示例, 并且运行它。



Processing 编程也能够全屏显示(在 Processing 中被称为“present mode”)你能够在菜单选项中找到: Sketch → Present(或者是 shift+点击 run)。如果你想草图覆盖整个屏幕你就必须设置你的草图为你屏幕的尺寸size()。

2.4 速写本

Processing的编程文件被简称为sketch(草图),在这本书中我们将持续应用这个叫法。存储你草图文件的文件夹我们称为你的sketchbook(速写本)。从技术上来说,当你在Porcessing中运行一个草图是,它会作为你电脑的一个本地应用。在本章和18章中我们都能看到Processing同样也允许你将你的草图导出为web格式(作为小程序嵌入到浏览器中运行)或其他的特定平台应用格式。

一旦你确认了 Processing 正常工作,你就可以准备看时创建你自己的草图了。点击“新建”按钮,会产生一个按日期命名的新的草图。另存为你的草图名称是一个不错的选择。(注: Processing 不允许名称中有空格或者连字符,并且你的草图名称不能以数字开头)

当你第一次运行Processing的时候，默认情况下Processing会默认将草图存储在“My Documents”（在Windos下）以及”Documents”（在Mac下）。当然你可以随意选择你硬盘上的任何位置进行存储，这只是一个默认文件夹。并且你能在Processing中的设置中改变存储位置（文件菜单栏中可以看到）。

每一个Processing文件包括了一个文件夹（和草图相同名字）以及“.pde”后缀的草图文件。假如你的Processing文件名为MyFirstProgram，那么你的文件夹也会是MyFirstProgram并且附带一个MyFirstProgram.pde文件。“.pde”是一个纯文本文件，其中包含源代码（之后我们会看到Processing文件能够有多个pde，但是现在智慧出现一个）。有一些Processing文件也会包括一个”data”名称的文件夹，里面会包含一些你使用的媒体元素，如图片、声音文件等。



练习2-3：在一个空白的草图中写入一些第一章所学的内容。注意某些文字的颜色。并运行草图。是你想的那样吗？

2.4 Processing 代码

现在终于可以开始编写代码了，可以使用第一章所学的一些元素进行讨论。让我们来看一些基本的与法规则。这里有三种不同的陈述书写方式：

- 函数调用
- 赋值操作
- 管理结构

就目前而言，每一行的代码就是一个函数的调用，如图2.4所示。我们将在以后的章节中探讨其他两个类别的书写。函数有一个名称，后面的括号中的内容是它们的参数。回顾第一章我们使用了函数去描述一个形状（我们只是叫他们“命令”或者“说明”）。所以我们可以将函数名称看作一个动词（画），参数看作一个对象。每个函数调用必须以分号结尾，如图2.5。

图 2.4



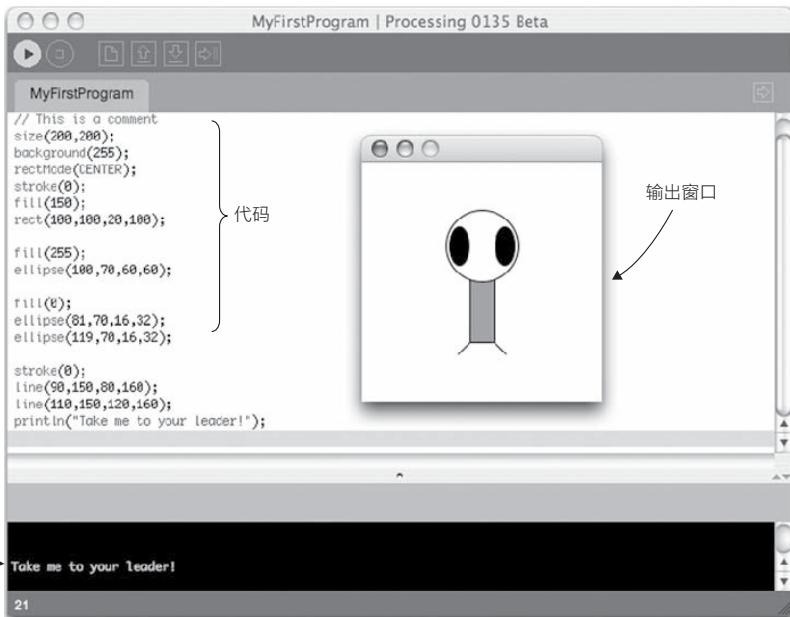
我们已经学会了一些函数，包括 background(), stroke(), fill(), noFill(), noStroke(), point(), line(), rect(), ellipse(), rectMode(), ellipseMode()。Processing会一个接一个的执行这些函数并且在窗口中显示出来。但是在第1章我们忘记学习了一个很重要的函数，size()。size()是窗口的指定的尺寸，并且有两个参数，宽和高。size()函数应该始终保持在第一位。

size(320,240);

打开一个窗口它宽度是 320px 和高度是 240px

让我们写第一个例子（如图 2.5 所示）。

图 2.5



有几个其它注意事项：

- Processing 文本编辑器会对已知词语着色（有时候也被称为“保留”字或“关键词”）。例如 Processing 库中的绘制函数，“内置”变量（我们会在第3章接触变量的概念）和常量，以及某些来自Java的编程语言。
- 有时候在 Processing 消息窗口中显示的信息是非常有用的（位于底部）。它可以用println()函数完成。println() 只要一个参数，一个字符串（我们会在第 14 章学到字符串）。当编程开始运行时，Processing 会显示这些字符串在消息窗口（如图2.5所示）并且在这里它会显示 "Take me to your leader!"，当你要试图调试变量值时（我们会在第 12 章学习调试）就会很有用了。
- 窗口左下角的数字代表现在你选择的代码行数。
- 你可以自己写一些“注释”在你的代码中。注释的文本 Processing 运行时会直接忽略掉。你能说明这些代码时什么意思，或者是提醒什么事。 // 两个斜杠代表的事单行注释。/* 代表多行注释，并且需要以 */结束。

```
// This is a comment on one line
/*
 * This is a comment that
 * spans several lines
 * of code */

```

在你编程中应该养成用短句注释的习惯。即使我们的草图非常简单和简短，你也应该写出一些注释。没有注释代码一般来说都很难阅读。你不需要注释每一行的代码，但是应该包括尽可能多的信息。注释也迫使你必须了解代码的工作流程。如果你不知道怎么工作怎么去写注释？

注释不会总是包括说明，因为在实际的编程中很多东西事不同的，在一本书中代码注释是很难读懂的。如果你不了解该怎么写，你可以看一下网站上我们书中的例子，注释始终都会出现。所以，我这么强调只是为了让你了解写注释的重要性！

```
//A comment about this code  
line(0,0,100,100);
```

代码的暗示



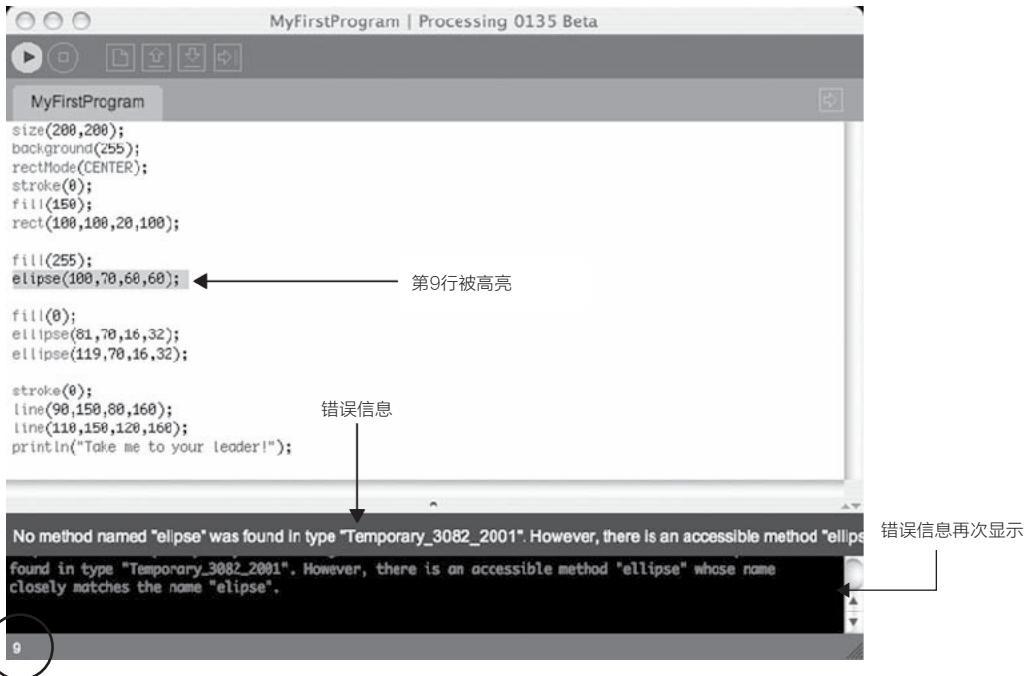
练习2-4：新建一个空白的草图。把你从第1章结尾所学到的代码输入到 Processing 的窗口中。并且添加注释来描述这些代码是做什么。然后添加一个 `println()` 语句在消息窗口显示。存储草图。点击“run”按钮，它是正常工作还是出现错误？

2.6 错误

在前面的例子中我们都是正确的代码，因为我们没有犯任何错误或者错别字。在程序员的生活中不犯错误是很罕见的。大多数时候我们第一次按“run”基本上都不能完美运行。当我们写错代码看看我们会发生什么，如图2.6。

图2.6显示了你打错了字，在第9行写成了“elipse”而不是“ellipse”。当你代码出现错误的时候点击运行时，Processing不会打开草图运行的窗口，并且会在消息窗口显示错误信息。这些消息很友好，它可能告诉我们可能我们有错别字“ellipse”。不是所有的Processing错误都显而易见的，在这本书中我们会继续寻找其他错误。还包括附录中的常见错误。

图 2.6



Processing 代码是区分大小写的！

如果你将 ellipse 写成 Ellipse，这绝对是错误的。

在上面例子中只有一处错误。如果你有多个错误，Processing 只会提醒你第一个错误（以此类推，一旦这个错误修改了，下一个错误会在再次运行时显示）。这虽然是一个很麻烦的限制，但是它往往有益于修复编程。

实际上这只会进一步强调开发的重要性就如我们书中介绍的。如果我们只在一个时间实现一个功能，那么我们就只能在一个时间出现一次错误。

练习2-5：尝试书写错误的语句，看看消息窗口是你期望显示的信息吗？





练习2-6：修改下面错误的代码。

```

size(200,200); _____
background(); _____
stroke 255; _____
fill(150) _____
rectMode(center); _____
rect(100,100,50); _____

```

2.7 Processing 参考

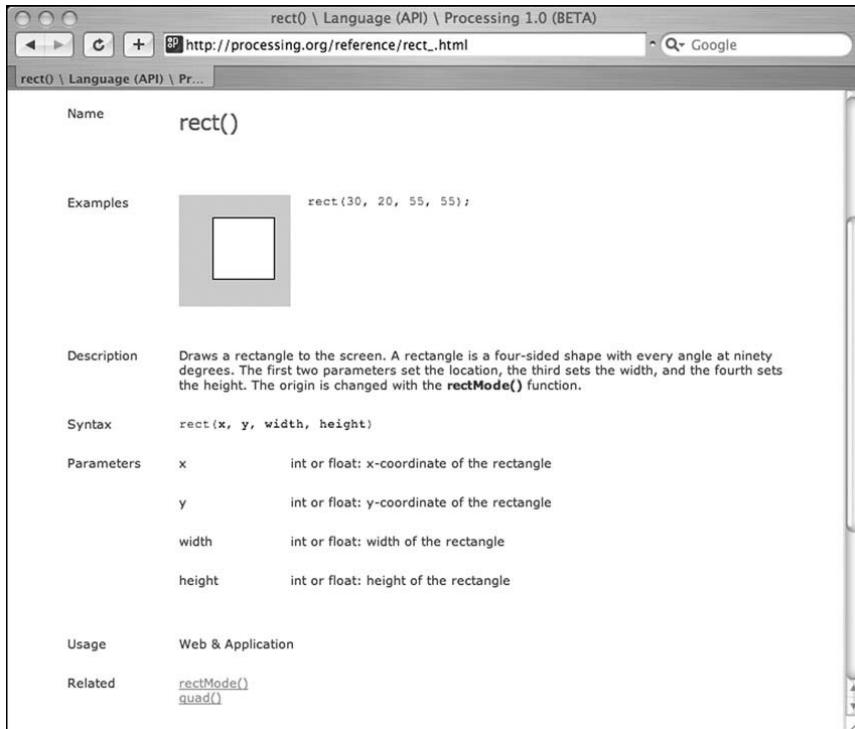
我们已经学习了一些简单的函数— `ellipse()`, `line()`, `stroke()` 等等，它们都是 Processing 库中的一部分。我们怎么能知道“`ellipse`”是对的而“`elipse`”是错的，或者我们怎么能知道`rect()` 中需要 4 个参数（x 坐标，y 坐标，宽度，高度）？这些都是通过阅读我们的编程在线参考来学习的。虽然我们能在这本书中学到很多元素，但是这不代表着这本书就能代替 Processing 参考。如果有问题请查看 Processing 参考。

Processing 参考你能在它的官方网站上找到 (<http://www.processing.org>)，在“reference”链接下。你能按类别浏览或者按字母顺序浏览。如果你查找 `rect()`，你就会看到它的所有解释。如图 2.7 所示。

正如你所看到的，参考页面提供 `rect()` 完整的函数释义，包括：

- **名称** — 函数名称。
- **示例** — 代码的举例。
- **说明** — 清晰地说明这个函数是做什么的。
- **语法** — 函数的正确书写语法。
- **参数** — 参数是括号中的元素。它会告诉你应该在括号中写入什么样的数据（数字，字符，或者其他等）以及它们的含义。（在之后的章节我们会深入了解）
- **返回** — 有时候你在调用一些函数时它会回馈你某些信息（例如两个数字相加，它会返回给你答案）。同样，在之后的章节我们会深入了解。
- **用法** — 有一些函数程序你能在网站上，但有的只能在 Processing 中本地运行。
- **相关方法** — 一个函数列表经常需要结合当前的函数。需要注意的是在 Java 中“functions（函数）”通常被称为“methods（方法）”。更多请参考第6章。

图 2.7



Processing同样也有一个非常简单的“查找参考”选项。双击你需要查找的关键字，然后点击HELP→ FIND IN REFERENCE（或按下Shift+Ctrl+F）。



练习2-7：使用 Processing 参考，查找我们尚未学过的函数，单要在“Shape”和“Color”类别范围内。



练习2-8：使用 Processing 参考，找到一个能改变你线的粗度的函数。看看它需要什么样的参数？并且绘制出一条 1px 粗度的线，然后再绘制出一条 5px 粗度的线，最后再绘制一条 10px 粗度的线。

2.8 “播放”按钮

Pecessing的好处之一就是它能点击“播放”按钮让程序动起来。假设我们有一个关于动画，电影，音乐或其他媒体的好点子，都可以运行。Processing输出的媒体是实时的计算机图形，所以为什么不让它们动起来呢？

尽管如此，但是需要我们考虑到的是我们再这里做的不是像ipod一样直接播放它们。Processing是将代码翻译成机器代码，然后才会执行播放。所有的步骤都是按照事先的顺序播放的。让我们来详细了解一下这些步骤，看看Processing是如何辛勤工作的吧。

- 步骤1** 转换成Java。Processing是真正的Java(在第23章中我们会有详细讨论)。为了你的代码能在计算机上运行，首先必须要将我们的语言转换成Java代码。
- 步骤2** 编译成Java字节代码。在第一步中我们只是将文件转换成另一个文件(Java格式的扩展文件，而不是pde)。为了计算机能够理解它，它还需要被翻译成计算机语言。这个过程被称为编译。如果你的变成是一个不同的语言，如C语言，这些代码会直编译成计算机语言。在Java中，代码会转换成一种特殊的计算机语言(Java字节代码)。它能够在不同的平台运行(Mac,Windows,手机，PDA中等等)只需要那些机器运行了“Java虚拟机”。虽然有时候这会导致它们可能会运行的有些慢，但是跨平台是Java的一大特点。欲了解更多有关内容，请访问<http://java.sun.com>或者考虑买一本Java编程的教程(在你看完这本书后)。
- 步骤4** 执行。编译成JAR文件后，JAR文件是Java的归档文件，其中包含已经编译的Java程序、图像、字体以及其他数据。然后由Java虚拟机执行JAR文件，并且显示在新窗口中。

2.9 你的第一个草图

现在我们已经下载并安装了Processing，而且还了解了记本的菜单和界面，熟悉了在线参考，所以我们编程的准备工作都做好了。正如我在第1章提到的，这本书上半部分将从一个例子开始说明编程的元素：变量、数组、条件语句、循环语句、函数以及对象。其他的例子也会随之出现，单这个例子会作为一个基本元素。

例子将按照我们的新朋友Zoog的故事展开，开始用简单的形状进行渲染。Zoog的发展故事将包括鼠标交互、运动以及克隆多个Zoog。如果你用自己的元素来进行设计，我建议你自己做出来。如图2.8所示。

例 1-5：展示了我制作的 Zoog，如图 1.21 所示。

```
size(200,200); // Set the size of the window
background(255); // Draw a black background
smooth();
```

smooth() 函数是开启“抗锯齿”功能，它能使我们图形的边缘显得平滑。如果没有**smooth()**那么抗锯齿功能就是关闭的。

```
// Set ellipses and rects to CENTER mode
ellipseMode(CENTER);
rectMode(CENTER);
```

```
// Draw Zoog's body
stroke(0);
fill(150);
rect(100,100,20,100);
```

Zoog 的身体

```
// Draw Zoog's head
fill(255);
ellipse(100,70,60,60);
```

Zoog 的头部

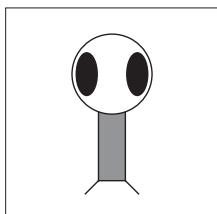
```
// Draw Zoog's eyes
fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
```

Zoog 的眼睛

```
// Draw Zoog's legs
stroke(0);
line(90,150,80,160);
line(110,150,120,160);
```

Zoog 的腿

图 2.8



写完代码之后我们就迫不及待的想在电脑屏幕上显示看看！（是的，我知道这很有意思）如果你需要运行本书中的任何代码示例，你可以有两个选择：

- 手动输入这些代码
- 访问本书的网站(<http://www.learningprocessing.com>)，找到例子的编号，复制/粘贴代码（或下载）。

第二个选项肯定相对来说更简单。但是在你实际学习中，真正有价值的还是你自己输入代码，这样你的大脑在你输入编程的时候会记住，并且记牢，你会学到很多。

当你准备复制/粘贴这些代码的时候你最好对这些代码是了解的。如果你开始的时候不了解它们的工作流程，最好还是用手打代码。



练习2-9：使用你在第1章的设计，事先你自己的屏幕绘图，只使用2D的原始形状—arc(), curve(), ellipse(), line(), point(), quad(), rect(), triangle()—以及记本的颜色函数—background(), colorMode(), fill(), noFill(), noStroke(), and stroke()。记住使用size()来规定你的窗口尺寸。建议：在每写一行代码的时候播放一次草图，沿途修正任何可能发生的错误。

2.10 你的第一个草图

当你完成了一个 Processing 草图时，你能够把它作为一个Java小程序发布在网上。一旦它是互教或者动画程序，它会很有趣。只要你完成了练习2-9并且确定它能够完美运行播放，选择FILE → EXPORT。

请注意如果你的程序中有错误，那么导出之后也不能正常运行，所以在导出之前请先测试。

在草图的文件夹中会创建一个新的目录名为“applet”的文件夹，如图2.9所示。

图 2.9

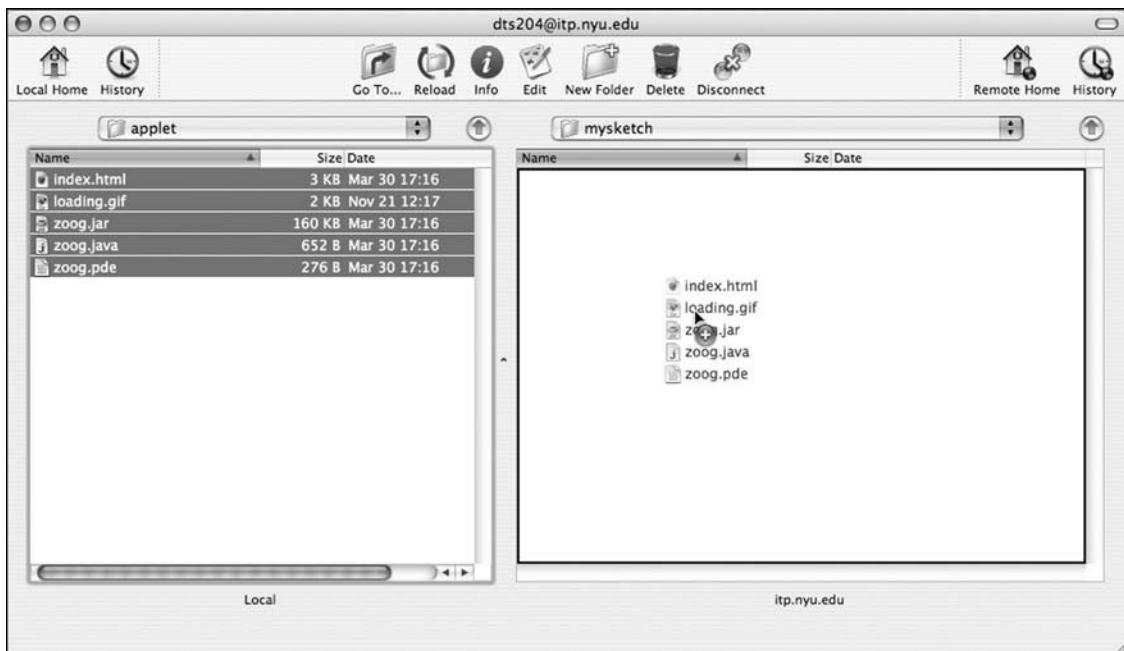


现在你有把你的完稿发布到网上必须的所有文件：

- **index.html** — 一个显示应用的HTML源代码
- **loading.gif** — 用户在加载这个小程序的时候的加载图片（ Processing会提供一个默认的，但你也可以自己创建一个）
- **zoog.jar** — 经过编译的小程序代码
- **zoog.java** — 经过翻译的Java源代码（看起来像你的Processing代码，但有一些Java自己额外的内容，详细参考第20章）
- **zoog.pde** — 你自己的Processing源代码文件

双击“index.html”文件能够在默认浏览器中运行这个小程序。如图2.10所示。要在线显示这些小程序，你需要网络服务器和FTP软件（或者你也可以使用Processing分享网站，如<http://www.openprocessing.org>）。你能在书的网站开始找到相关内容。

图 2.10



练习2-10：导出你所制作的小程序并在浏览器中运行播放（无论是本地还是上传的）。



3. 互动

“一定要记住，这整个神奇的事情都是从一个理想和一个鼠标开始的。”

— 沃尔特·迪士尼

本章内容：

- 让程序流动起来
- `setup()` 和 `draw()` 的含义
- 鼠标交互
- 你的第一个动态 Processing 编程
- 处理事件，如鼠标点击以及键盘按键

3.1 流动起来

如果你曾经玩过一个电脑游戏，他是互动的数字艺术，你会发现这很神奇。游戏开始时你从邪恶的领主xxx中拯救公主，达到了高分，然后游戏结束。

我想把随着事件推移而流动的东西集中在这一章。一个游戏开始你需要设置一些出示条件：你的名字，你的性格，你从 0 分开始，并且你是等级1。我们认为这部分就是编程的设置。初始化这些条件后你就可以开始玩游戏了。在每一个时间点，计算机都会检查你用鼠标在做什么，计算所有的游戏任务的行为并利用显卡显示在屏幕上。在这些过程中，计算和绘制一遍又一遍的发生着，理想情况流畅的动画应该是每秒30次。这就是程序的绘制。

这个概念对我们来说在 Processing 中摆脱贫出静态设计（第二章）是非常重要的。

步骤1 在编程第一时间设置初始条件。

步骤2 做一些不断重复的事情，直到程序退出。

思考一下你可能会怎么进行跑步比赛。

步骤1 穿上你的运动鞋并且做伸展运动，这只需要做一次，对不对？

步骤2 先迈出右脚再迈出左脚，不断重复，尽可能快的。

步骤3 经过 26m 的时候，退出。

练习3-1：在英语中写一个简单的“流”，就像Pong。如果你不了解Pong，请访问：<http://en.wikipedia.org/wiki/Pong>。



3.2 我们的好朋友 `setup()` 和 `draw()`

为了更好的学习编程，我们可以利用这个新发现的概念，将它应用到我们的第一个“动态” Processing 草图。与第2章的静态示例不同，这些编程将是连续性的绘制到屏幕上的（除非你退出）。这由2个代码块组成 `setup()` 和 `draw()`。之后的章节我们会深入讨论写我们自己的函数；但是现在我们只需要知道这2个部分。

什么是代码块？

代码块表示的是被大括号括起来的所有代码。

```
{
  A block of code
}
```

代码块可以相互嵌套。

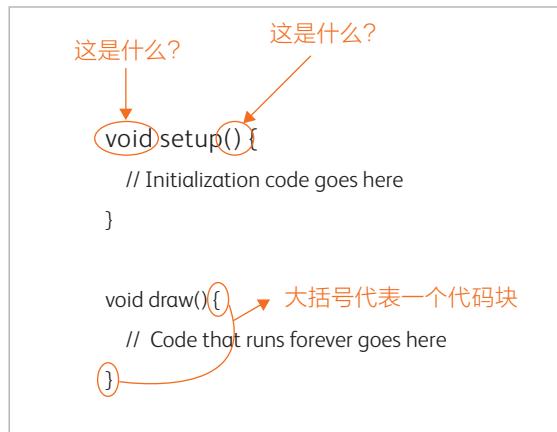
```
{
  A block of code
  {
    A block inside a block of code
  }
}
```

代码块是非常重要的，因为它让我们分开大的代码部分，管理它们。我们编程由一个惯例，为了使每个区块内为了代码更易读我们对进行代码缩进。Processing 中可以通过 Auto-Format option 自动输出这种排版方式（Tools → Auto-Format）。

代码块将揭示在编程开发过程中越来越复杂的逻辑，是至关重要的变量，条件语句，重复语句，对象以及函数，在之后的章节我们会深入讨论。现在只需要看2个简单的代码块：`setup()` 和 `draw()`。

让我们来看看 `setup()` 和 `draw()`，你一定觉得前面那些代码很奇怪，如图3.1

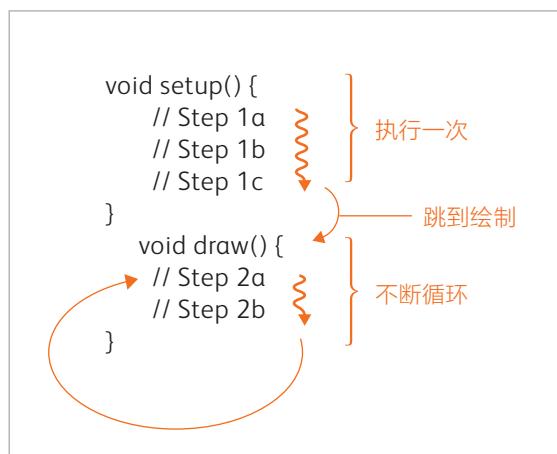
图 3.1



诚然，在图 3.1 中看起来我们还没有做好完全准备去了解很多东西。我们已经知道大括号内部表示的是代码块，但是大括号为什么在“`setup`”和“`draw`”的后面呢？天哪，还有“`void`”是什么意思？不知道这一切是很自然的，这些语法在以后的章节会变得越来越重要。

对于现在来说，我们的关键是了解图 3.1 对于“流”的结构控制。如图 3.2 所示。

图 3.2



它是如何运行的？当我们开始运行编程，它会精确的按照我们的指示，第一步执行 `setup()`，然后继续向下执行 `draw()`。为了表示的更清楚，就像如下：

1a,1b,1c,2a,2b,2a,2b,2a,2b,2a,2b,2a,2b ...

它是如何运行的？当我们开始运行编程，它会精确的按照我们的指示，第一步执行setup()，然后继续向下执行draw()。为了表示的更清楚，就像如下：

例 3-1：动态Zoog草图

```
void setup(){
    // Set the size of the window
    size(200,200);
}

void draw() {
    // Draw a white background
    background(255);

    // Set CENTER mode
    ellipseMode(CENTER);
    rectMode(CENTER);

    // Draw Zoog's body
    stroke(0);
    fill(150);
    rect(100,100,20,100);

    // Draw Zoog's head
    stroke(0);
    fill(255);
    ellipse(100,70,60,60);

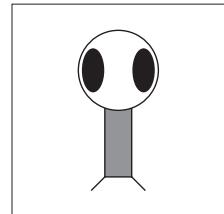
    // Draw Zoog's eyes
    fill(0);
    ellipse(81,70,16,32);
    ellipse(119,70,16,32);

    // Draw Zoog's legs
    stroke(0);
    line(90,150,80,160);
    line(110,150,120,160);
}
```

setup()会在第一时间运行。**size()** 应该事中保持在第一行的位置，因为Processing 在没有指定窗口时无法做任何事。

draw() 会不断循环，除非你把运行的草图关闭。

图 3.3



写出例3-1的代码并运行它。你会觉得很奇怪对吧？它在窗口没有任何变化。这看起来和静态的草图是一样的！这是怎么回事？

好吧，如果我们检查代码我们会注意到 draw() 函数没有任何变化。每次循环都会执行相同的指令。所以，随着时间的推移，它只是在重复相同的绘制，所以它看起来和我们的静态编程一样。



练习3-2：重新绘制出你在第2章绘制的动态编程。虽然它看起来没变化，但是很有成就感！

3.3 鼠标交互

思考一下，如果你在绘制函数中输入的不是精确的数字参数，而是“鼠标的x轴位置”或者“鼠标的y轴位置”，那会是怎样？

```
line(the mouse's X location, the mouse's Y location, 100, 100);
```

实际上，你可以不用具体的数字来描述参数，但是你必须使用关键词mouseX和mouseY,它表示的是你鼠标的水平位置以及垂直位置。

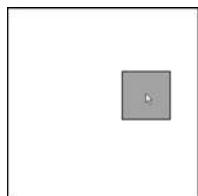
例 3-1：动态Zoog草图

```
void setup() {
  size(200,200);
}
void draw() {
background(255);
// Body
stroke(0);
fill(175);
rectMode(CENTER);
rect(mouseX,mouseY,50,50);
}
```

试一试将background()放在setup()中。(练习 3-3)

mouseX是一个关键词，它表示在草图中鼠标现在所在的水平位置。mouseY是一个关键词，它表示在草图中鼠标现在所在的垂直位置。

图 3.4



练习3-3：请解释一下为什么我们将background()从draw()移动到setup()中时，长方形移动的轨迹还会留下来。



一行隐形的代码

如果你遵循编程的顺序 setup() 紧贴着 draw()，你可能会得出一个有趣的问题：什么时候 Processing 会真实的显示在窗口中？什么显示的是新的？

乍一看，人们可能会认为每次重复都会包括每一行代码包括drawing函数。然而如果是这样，那么我们将看到的是在一个时间出现一个形状显示。这样发生的频率很快，我们就很难注意到每个形状单独出现的状态。但是当窗口每次被清除的时候background()会被调用，有一个不好的结果是会发生闪烁。

Processing解决了这个在draw()中结束后，重复更新窗口的问题。这是因为在每次结束时有一种隐形的代码会渲染窗口。

```
void draw() {
    // All of your code
    // Update Display Window -- invisible line of code we don't see
}
```

这个过程被称为双缓冲处理，并且在一个较低水平的编程环境中你会发现你必须自己实现这个功能。同样，我们很感谢Processing的开发者，让我们可以更简单的介绍这个软件。

这个过程被称为双缓冲处理，并且在一个较低水平的编程环境中你会发现你必须自己实现这个功能。同样，我们很感谢 Processing 的开发者，让我们可以更简单的介绍这个软件。

我们可以进一步推动这个想法和创建一个被 mouseX 以及 mouseY 控制的更复杂的图案（多种形状和多种颜色）。例如，我们能重新改写 Zoog，让它跟着鼠标动。注意Zoog的身体是一个确切的位置(mouseX, mouseY)，而其他的身體部分是相对于鼠标的位置。例如 Zoog 的头，位置在 (mouseX, mouseY-30)。按照下面的例子，仅移动 Zoog 的身体以及头部，如图3.5所示。

例 3-3：Zoog的动态变化草图

```
void setup() {
size(200,200); // Set the size of the window
smooth();
}

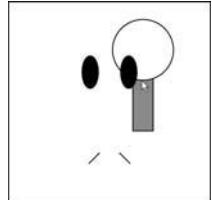
void draw() {
background(255); // Draw a white background

// Set ellipses and rects to CENTER mode
ellipseMode(CENTER);
rectMode(CENTER);

// Draw Zoog's body
stroke(0);
fill(175);
rect(mouseX,mouseY,20,100);

// Draw Zoog's head
stroke(0);
fill(255);
ellipse(mouseX,mouseY-30,60,60);
```

图 3.5



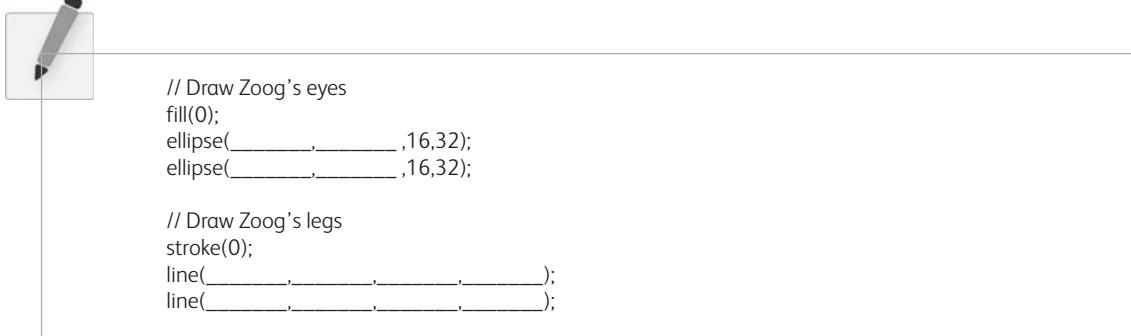
Zoog 身体的位置是 (mouseX, mouseY).

Zoog 头部在身体上方，位置是 (mouseX, mouseY-30).

```
// Draw Zoog's eyes
fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
// Draw Zoog's legs

stroke(0);
line(90,150,80,160);
line(110,150,120,160);
}
```

练习3-4：让 Zoog 除了身体以外的部位通过鼠标移动起来。

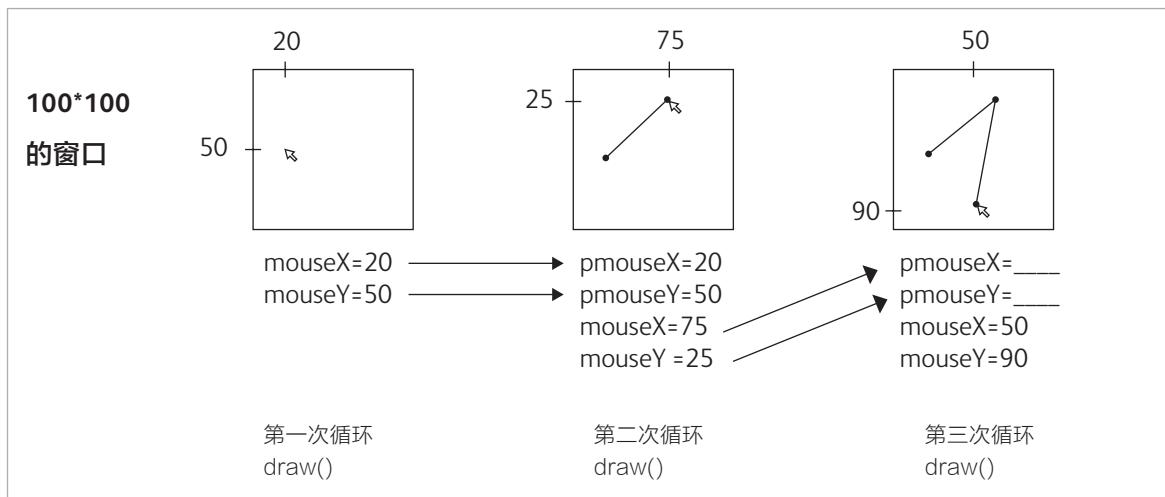


练习3-5：重新编辑鼠标的响应（用不同的颜色的位置）。



除了 mouseX 和 mouseY 以外，你还可以使用 pmouseX 和 pmouseY。这2个关键词代表着“先前的” mouseX 和 mouseY 的位置，那就是鼠标最近一次 draw() 的循环。这让一些有趣的互动变成可能。例如，让我们思考一下能发生什么，如果我们从以前的鼠标位置画一条线到当前鼠标的位置，如图3.6所示。

图 3.6





练习3-6：填写图3.6中的空白。

通过连接先前和现在的鼠标位置，draw()上的线条络绎不绝的出现。如图3.7所示。

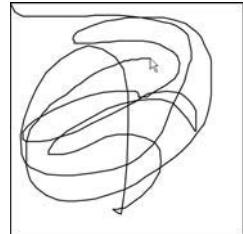
例 3-4：Zoog的动态变化草图

```
void setup() {
    size(200,200);
    background(255);
    smooth();
}

void draw() {
    stroke(0);
    line(pmouseX,pmouseY,mouseX,mouseY);
}
```

从先前的位置到现在的位置
 画一条线

图 3.7



练习3-7：计算鼠标水平方向 `mouseX` 和 `pmouseX` 之间的差异的绝对值。一个数的绝对值没有前面的符号，它被定义为：

- -2的绝对值是2。
- 2的绝对值是2。

在Processing中，如果我们需要取得一个数字的绝对值，那我们就需要把那个数字放到abs()函数中，就像这样

- `abs(5) → 5`

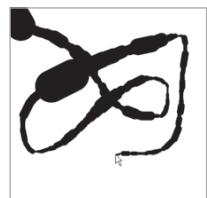
鼠标的移动速度就如下：

- `abs(mouseX-pmouseX)`

更新练习3-7，让用户更快的移动鼠标，绘制更有意思的线。提示：可以在参考中查找strokeWeight()。

```
stroke(255);
_____(_____);  

line(pmouseX,pmouseY,mouseX,mouseY);
```



3.3 鼠标点击和按键操作

当我们通过setup()和draw()框架以及mouseX, mouseY关键词创作生动有趣的交互草图时，却发现一些关键的互动形式却没有一鼠标点击！

为了学习怎么通过鼠标点击发生一些交互。我们需要回到我们现在学到的“流”。我们知道setup()会运行一次，draw()会不断重复循环。那么鼠标点击呢？鼠标按下（以及键盘按下）在Processing中会被作为一个事件。如果我们想让在鼠标点击时发生一些有趣的事（如改变背景的颜色），我们就需要添加第3个代码块来处理这个事件。

这个事件的“函数”将会告诉我们当事件发生的时候会执行哪些代码。setup()里面的代码会出现一次，并且只出现一次，即，对于每个发生的事件只运行一次。一个事件，如鼠标点击，可以发生多次！

这里是我们需要的2个新函数：

- mousePressed() — 负责鼠标点击
- keyPressed() — 负责按键操作

下面的示例用到了这2个事件函数，当鼠标点击的时候就会增加正方形，当按键时就会清除背景。

例 3-5：mousePressed() 和 keyPressed()

```
void setup() {
    size(200,200);
    background(255);
}

void draw() {
```

在这个例子中draw()什么都没有

```
}

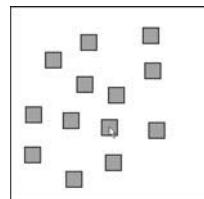
void mousePressed() {
    stroke(0);
    fill(175);
    rectMode(CENTER);
    rect(mouseX,mouseY,16,16);
}

void keyPressed() {
    background(255);
}
```

每当用户点击鼠标，mousePressed()里面写的代码就会被执行

每当用户按键，keyPressed()里面写的代码就会被执行

图 3.8



在例3-5中，我们有4个函数在变成流程里面。首先从 setup() 开始初始化尺寸以及背景颜色，然后继续到draw() 重复循环。因为 draw() 部包含人和代码，所以创考将保持空白。但是我们添加了2个新的函数：mousePressed() 和 keyPressed()。当用户点击鼠标（或按下一个键），就会执行一次他们各自代码块中的代码。



练习3-8：添加“background(255);”到 draw() 函数中，为什么程序就停止工作了？

现在我们准备把Zoog的元素组合在一起。

- Zoog的全身都会跟着鼠标
- Zoog眼睛的颜色会取决于鼠标的位置
- Zoog的腿会是从鼠标之前的位置到现在的位置的长度。
- 当你鼠标点击时，在窗口会显示一些信息：“Take me to your leader!”

注意在例3 - 6有额外的函数frameRate()。它需要1到60之间的整数，他会强制Processing中draw()循环的速度。如 frameRate (30) 代表每秒30帧，这是计算机动画的传统速度。如果你在Processing中没有写 frameRate()，那么Processing的草图会每秒60帧运行。由于每台电脑的效率不同，为了确保你的草图在所有计算机中的速度是相同的，请写入frameRate()。

这个帧速率仅仅是一个最大流畅度的速率。如果你绘制100万个矩形，可能它的运行速度就没这么快了。

例 3-6：互动的Zoog

```
void setup() {
    // Set the size of the window
    size(200,200);
    smooth();
    frameRate(30);
}

void draw() {
    // Draw a black background
    background(255);

    // Set ellipses and rects to CENTER mode
    ellipseMode(CENTER);
    rectMode(CENTER);

    // Draw Zoog's body
    stroke(0);
    fill(175);
    rect(mouseX,mouseY,20,100);

    // Draw Zoog's head
    stroke(0);
```

运行帧数是每秒30帧

```
// Draw Zoog's eyes  
fill(mouseX,0,mouseY);  
ellipse(mouseX-19,mouseY-30,16,32);  
ellipse(mouseX + 19,mouseY-30,16,32);  
  
// Draw Zoog's legs  
stroke(0);  
line(mouseX-10,mouseY + 50,pmouseX-10,pmouseY + 60);  
line(mouseX + 10,mouseY + 50,pmouseX + 10,pmouseY + 60);  
}  
  
void mousePressed() {  
    println("Take me to your leader!");  
}
```

眼睛的颜色会取决于鼠标的位置

腿会是从鼠标之前的位置到现在的位置的长度



课程一项目

(通过第1 - 3章的练习你可能已经完成了这个项目。这个项目就是把所学的东西都放在一起。希望你从头开始使用这些元素进行设计。)

步骤1 设置静态屏幕绘制，并且使用RGB色彩以及基本图形。

步骤2 使这个静态屏幕通过鼠标交互变的动态起来。让这些形状跟着鼠标动，并且通过鼠标的移动改变他们的颜色等等。

第二课

你需要了解的一切

- 4. 变量
- 5. 条件
- 6. 循环

4. 变量

“全世界所有的书加起来的信息比美国城市一年播放的视频信息还大。但是不代表它们价值相同。”

— 卡尔·萨根 (Carl Sagan)

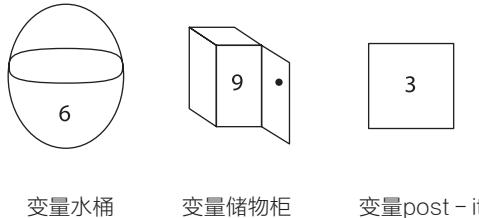
本章内容：

- 变量：它们是什么？
- 声明和初始化变量
- 变量的常见用途
- “免费”获取变量（又称“内置”变量）
- 为变量使用随机值

4.1 什么是变量

我承认一点，当我在教编程时，我会试图用类比的方式去解释变量的概念。我可能会说“变量像一个水桶”当你有灵感的时候你就会把东西给放进去。“变量像一个储物柜”，可以存放一些心血，并且可以随手可得。

图 4.1



我可以这样继续讲下去，但我不会。我认为你差不多了解含义，但是我不能完全肯定你是否完全理解。这里能解决它。

计算机有内存，但是为什么我们叫它内存？因为它要记住电脑使用的东西。

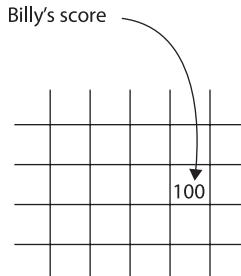
从技术上讲，一个变量的命名来自于计算机中内存储存的数据地址（“内存地址”）。由于计算机在同一个时间只能处理一次命令，变量允许程序员存储编程每一次信息点并且在之后能拿回来使用。对于Processing程序员，变量非常有用；变量能够很精确的指出一做一个三角形从蓝色到紫色，并且在屏幕上飞一圈收缩称矩形的这些动作。

在所有关于变量的类比中，我更倾向于选择一张纸：方格纸。

试想一下，电脑的内存时一张方格纸，方格纸的每个单元都有一个地址。通过我们对像素的学习，我们学会了如何用列和行对方格纸中的位置进行描述。同样，有了变量，我们也能这样描述。

让我们叫它“Billy’s Score”（在下一节中我们会知道为什么这样叫），并且给他一个值100。这样以来，当我们要在编程中使用它的时候我们并不需要记住它的值。它在内存中，我们可以使用他的名称。如图4.2所示。

图 4.2



变量的能力不仅仅是记住某个值。变量的整个点都可以变化，我们可以定期改变它的值，让它变的更有趣。

想想 Billy 和 Jane 之间的拼字游戏。为了跟踪分数，Jane拿出纸和笔，并记下2列名字：“Billy’s Score”和“Jane’s Score”。如果我们想象这个游戏是计算机上的虚拟游戏，我们突然能看到变量可以变化的概念。那张纸是电脑的内存，，在纸上写的信息 — “Billy’s Score” 和 “Jane’s Score” 是变量，在内存中，每一个玩家的总分都会随着时间的推移而发生变化。如图4.3所示。

图 4.3

Jane’s Score	Billy’s Score
5	10
30	25
53	47
65	68
87	91
101	98

在我们的拼字游戏的例子中，变量有2个元素 — 一个是名称（如“Jane’s Score”），一个是值（如101）。在Processing中，变量可以容纳不同类型的值，在我们使用变量之前，我们需要明确定义这个值的类型。

练习4-1：思考一下Pong这个游戏，在编程游戏中你会需要什么样的变量？（如果你不熟悉Pong，参考<http://en.wikipedia.org/wiki/Pong>）。



4.2 变量的声明以及初始化

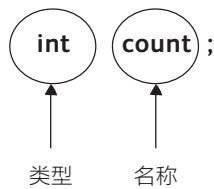
变量可以保存原始的值，或者引用对象和数组。对于现在，我们只用关心原始值 – 我们会在之后的章节学习对象和数组。院原始值构建了计算机上的数据，并且通常是一个单独的信息，如一个数字或者字符。

声明变量首先要说明类型，然后才是名称。变量名必须是一个单词（没有空格）并且必须以字母开头（它可以包含数字，但不能以数字开头）。它不能包含任何标点符号或者特殊字符，下划线“_”除外。

A类型是存储变量数据的一种。它可能是一个整数，一个十进制数，或者一个字符。这里的数据类型是你经常会使用的：

- 整数，如0、1、2、3、-1、-2等等，它们被称为“integers”，并且它们的简称关键词是“int”。
- 十进制数，如3.14159、2.5以及-9.95等等，它们被称为“floating point values”，并且它们的简称关键词是“float”。
- 字符，如字母‘a’、‘b’、‘c’等等，它们在变量类型中被称为“char”，并且在声明中如座位一个字母的时候需要用单引号括起来就像‘a’。在你想要申明键盘上的按键时字符很有用，如果涉及到其他的可以用文本字符串（参考第17章）。

图 4.4



如图4.4,我们声明了一个“int”类型的叫“count”的变量，他代表整数类型的变量。其他的数据类型如下表。

不要忘记

变量必须有一个类型。为什么？这时为了让电脑知道需要用多少内存分配给这个变量数据。

变量必须有一个名称。

所有的基本类型

- boolean: 对或错
- char: 一个字符, ‘a’、‘b’、‘c’ 等等
- byte: 一个小的数字, -128 到 127 之间
- short: 一个较大的数字, -32768 到 32767 之间
- int: 一个很大的数字, -2147483648 到 2147483647 之间
- long: 一个非常巨大的数字
- float: 一个十进制数, 如3.14159等等
- double: 一个小数点后更多数字的十进制数字（仅仅只是在高级编程或者对精度要求很高的时候使用）

一旦声明了变量，我们就可以将其设定一个与其设置相等的东西。在大多数情况下，如果我们忘记给变量初始化设置，Processing会给予它一个默认值，如整数就是0或者浮点数就是0.0等等。但是在声明完之后进行初始化设置时一个好的习惯，这样能够避免混淆一些数据。

```
int count;  
count = 50;
```

用2行代码声明以及初始化变量

为了更简洁，我们可以结合以上两个语句为一体。

```
int count = 50;
```

用1行代码声明以及初始化变量

关于变量命名的友情提示！

- 避免使用在Processing中其他地方可能会出现的单词。不要将你的变量名命名为mouseX，这个已经有了！
- 命名能够解释它的含义。这似乎是很显而易见的，但是它是一个重点。例如你正在用一个变量来追踪得分，就可以命名为“score”，而不是不相关的“cat”。
- 以小写字母开始命名书写，如果有其他单词跟着，后面跟着的单词首字母大写保留类（参考第8章）。如：“frogColor”是正确写法，“Frogcolor”不是正确写法。

变量也可以初始化其他变量值（如 $x = y$ ），或者通过数学计算的表达式（ $x=y+z$ 等等）下面是一些例子：

例 4-1：变量的声明以及初始化示例

```
int count = 0;           // Declare an int named count, assigned the value 0
char letter = 'a';       // Declare a char named letter, assigned the value 'a'
double d = 132.32;       // Declare a double named d, assigned the value 132.32
boolean happy = false;   // Declare a boolean named happy, assigned the value false
float x = 4.0;           // Declare a float named x, assigned the value 4.0
float y;                 // Declare a float named y (no assignment)
y = x + 5.2;             // Assign the value of x plus 5.2 to the previously declared y
float z = x*y + 15.0;    // Declare a variable named z, assign it the value which
                        // is x times y plus 15.0.
```



练习4-2：写出Pong这个游戏的变量值的声明以及初始化。

4.3 使用变量

虽然它在最初可能多数用在数字上，看起来比较复杂，但是变量会让我们的生活更轻松更有趣。

让我们举一个简单的例子，在屏幕上画一个圆。

现在，我们将在顶部添加变量

```
void setup() {
  size(200,200);
}

void draw() {
  background(255);
  stroke(0);
  fill(175);
  ellipse(100,100,50,50);
}
```

在第3章，我们学习了如何简单的利用mouseX, mouseY通过鼠标的位置分配形状尺寸以及位置。

```
ellipse(mouseX,mouseY,50,50);
```

你能看出来怎么回事吗？mouseX和mouseY被命名为鼠标的水平坐标和垂直坐标。它们是变量！然而因为它们本身在Porcessing环境中（当你输入它们时它们会显示红色），所以它们不用被声明。内置变量（又名“系统”变量），我们将在下一节进行进一步讨论。

我们现在要做的就是创建我们自己的变量声明和初始化，如上文所述，把变量纺织在我们代码的顶部。你也能在你代码的其他地方声明变量，我们稍后会介绍。现在，为了避免混淆，所有变量应该在顶部。

使用变量的经验法则

在什么时候使用变量我们没有硬性规定，但是如果你在代码值很多的情况下，很难阅读代码的情况下你就有必要将这些值转化成变量了。

有些程序员觉得一个数字出现3次或3次以上它就应该被转化成变量。但是就我个人而言，如果一个数字出现，我就转化成变量，始终使用变量。

例 4-2：变量的声明以及初始化示例

```
int circleX = 100;
int circleY = 100;

void setup(){
    size(200,200);
}

void draw(){
    background(100);
    stroke(255);
    fill(0);
    ellipse(circleX,circleY,50,50);
}
```

在顶部声明以及初始化一个整数
变量

用变量指定圆的位置

运行这段代码，你会发现它和第一个例子中得出的结果是一样的：在屏幕中间出现了一个圆。尽管如此，我们还是要提醒自己，变量不仅仅只代表一个值，我们把它称作变量因为他在变化。要改变它的值，我们需要写赋值操作，它会分配一个新的值。

到现在位置，我们写的每一行代码都叫做函数：line(), ellipse(), stroke()等等。变量中引入赋值操作能够进行组合。我们来看看这里像什么（它很像我们初始化变量的步骤，只有变量名称时不需要声明）。

variable name = expression

```
x = 5;
x = a + b;
x = y - 10 * 20;
x = x * 5;
```

用变量指定圆的位置

一个常见的例子就是递增。在上述代码中，circleX最初的值时100。如果我们想让circleX以1的速度递增，我们就说circleX等于它自身加1。在代码中，就是“circleX = circleX + 1;”。

让我们尝试加入到我们的编程中（让circleX的初始值为0）。

例 4-3：变化的变量

```
int circleX = 0;
int circleY = 100;

void setup() {
    size(200,200);
}

void draw() {
    background(255);
    stroke(0);
    fill(175);
    ellipse(circleX,circleY,50,50);
}
```

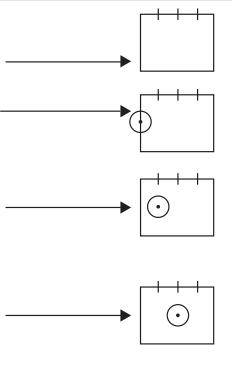
关于circleX一个以1递增的赋值操作。

}

这样会发生什么？如果你执行例4-3，你会注意到圆在从左至右移动。请记住draw()会不断的进行循环，并且同时把circleX的值保留在内存中。假设我们的电脑已经运行了一会。（这看起来很简单，但是这是我们理解变成动画的关键性原则。）

图 4.5

1. 记住 circleX = 0, circleY = 100
2. 运行 setup(). Open a window 200*200
3. 运行 draw().
 - 画圆 (circleX, circleY) → (0,100)
 - circleX 加 1
$$\text{circleX} = 0 + 1 = 1$$
4. 运行 draw().
 - 画圆 (circleX, circleY) → (1,100)
 - circleX 加 1
$$\text{circleX} = 1 + 1 = 2$$
5. 运行 draw().
 - 画圆 (circleX, circleY) → (2,100)
 - circleX 加 1
$$\text{circleX} = 2 + 1 = 3$$



练习如何遵循代码一步一步将带领你，你需要问的问题，然后写你自己的草图。成为与计算机之一。

- 你和你的电脑需要为草图记住什么的数据？
- 你和你的电脑怎么使用这些数据再电脑上绘制图形？
- 你和你的电脑怎样修改数据让草图的作品变的互动起来？

练习4-4：改变例4-3中圆从左至右的动画，让它变成圆从小大的动画。如果需要圆跟着鼠标增大应该改变什么内容？怎么改变圆的增长速度？



```
int circleSize = 0;
int circleX = 100;
int circleY = 100;

void setup() {
    size(200,200);
}

void draw() {
    background(0);
    stroke(255);
    fill(175);

    _____
    _____
}

}
```

4.4 多个变量

让我们进一步看看例子，并且对每一个数据信息使用变量。我们也可以使用浮点类型表示更精确的变量值。

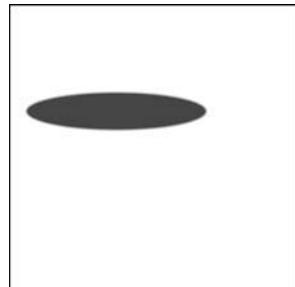
例 4-3：变化的变量

```
float circleX = 0;
float circleY = 0;
float circleW = 50;
float circleH = 100;
float circleStroke = 255;
float circleFill = 0;
float backgroundColor = 255;
float change = 0.5;

// Your basic setup
void setup() {
    size(200,200);
    smooth();
}
```

我们现在有8个变量！并且全部都是浮点类型！

图 4.6



```

void draw() {
    // Draw the background and the ellipse
    background(backgroundColor);
    stroke(circleStroke);
    fill(circleFill);
    ellipse(circleX,circleY,circleW,circleH);

    // Change the values of all variables
    circleX = circleX + change;
    circleY = circleY + change;
    circleW = circleW + change;
    circleH = circleH - change;
    circleStroke = circleStroke - change;
    circleFill = circleFill + change;
}

```

变量可以用在任何地方！背景，秒便，填充，位置，尺寸都可以！

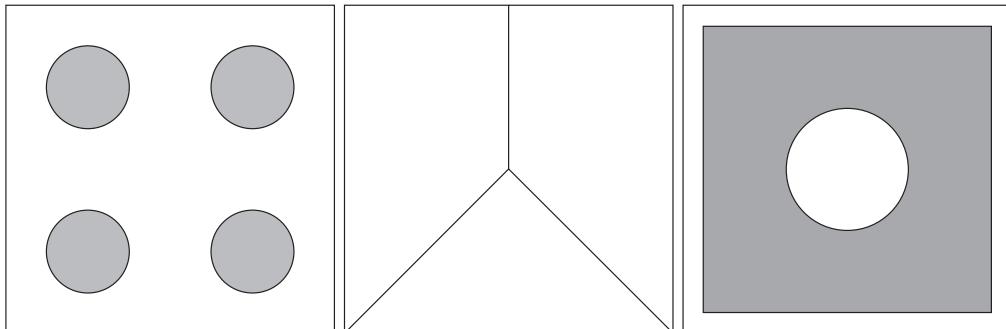
使用变量改变其他变量的递增和递减。

 练习4-4：改变例 4-3 中圆从左至右的动画，让它变成圆从小大的动画。如果需要圆跟着鼠标增大应该改变什么内容？怎么改变圆的增长速度？

步骤1 直接用普通代码写出下面截图中的样子（颜色随意使用）。

步骤2 用变量代替普通代码再写一次。

步骤3 再draw()中使用赋值来改变变量的值。如：“variable1 = variable1 + 2;”。试一试用其他的表达方式，看看会发生什么！



4.5 系统变量

正如我们看到的 mouseX 和 mouseY，Processing 会自带一些方便的系统变量供使用。这些通常是你在草图中需要的东西（如窗口的宽度，按键的上下键等等）。当你要自己声明变量时最好避免与系统变量重名，一旦你出现重名，你自己声明的变量会变成主要的那个，并且会覆盖之前的那一个系统变量。这里是一些常用的系统变量（还有更多，你可以在 Processing 参考中找到）。

- **width** — 草图窗口的宽度（像素）
- **height** — 草图窗口的高度（像素）
- **frameCount** — 运行帧的数量
- **frameRate** — 运行帧的速率（每秒）
- **screen.width** — 整个屏幕的宽度（像素）
- **screen.height** — 整个屏幕的高度（像素）
- **key** — 最近一次在键盘上按下的键
- **keyCode** — 键盘上的数字键
- **keyPressed** — 真或假？按下了哪个键吗？
- **mousePressed** — 真或假？按下了鼠标吗？
- **mouseButton** — 按下了那个按钮？左键，右键或者中间的键？

下面是一个例子，会利用上述的一些变量；我们没有全部使用到，因为以上的变量有的需要更高级的编程才能发挥功能。

例 4-5： 使用系统变量

```
void setup(){
    size(200,200);
    frameRate(30);
}

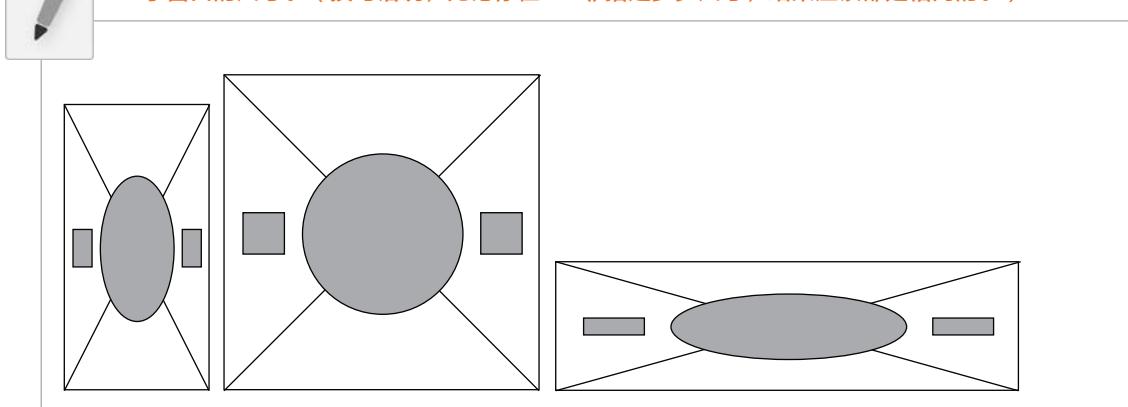
void draw(){
    background(100);
    stroke(255);
    fill(frameCount/2);
    rectMode(CENTER);
    rect(width/2,height/2,mouseX+10,mouseY+10);
}

void keyPressed(){
    println(key);
}
```

frameCount 用于给矩形上色

**如果我们把矩形的位置设置的是
(width/2, height/2) 时，那么它会始终
保持在窗口正中间**

练习4-5：使用 width 和 height ，重新创建下面的截图。这里需要主义：形状的调整必须是相对于窗口的尺寸。（换句话说，无论你在 size() 指定多少尺寸，结果应该都是相同的。）



4.6 随机：变化生活的调味品

你可能注意到现在的例子都有一点单调。一个圆圈在这，一个放行在那，浅灰色，另一个浅灰色。。

有一种方法能让你感觉很有意思。这一切都要追诉到这本书的背后：循序渐进的发展原理。只有我们学好了基础才更容易学其他的东西，一步一步来嘛。

尽管如此，我们已经耐心地通过了四个章节，并且现在时机已经成熟，我们就可以开始玩一些有意思的东西了。通过使用函数random() 我们会证明这一点。首先，让我们思考一下，例4-6，其输出如图4.7所示。

例 4-6：圆的变量

```
float r = 100;
float g = 150;
float b = 200;
float a = 200;
```

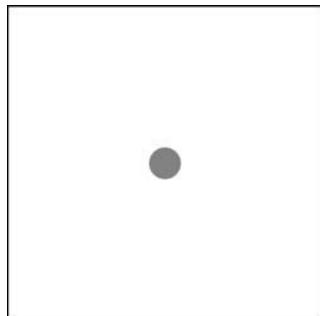
声明和初始化变量

```
float diam = 20;
float x = 100;
float y = 100;
void setup() {
    size(200,200);
    background(255);
    smooth();
}
```

```
void draw() {
    // Use those variables to draw an ellipse
    stroke(0);
    fill(r,g,b,a);
    ellipse(x,y,diam,diam);
```

使用这些变量（请记住颜色的第四个参数是透明度）。

图 4.7



图中就是我们那单调的圆圈。当我，我们可以调整它的变量值以及移动它的位置，增加它的尺寸，改变颜色等等。然而我们能不能每次都通过draw()，我们能做一个随机尺寸，颜色，位置的新的圆圈？random() 函数正是允许我们这样做。

random() 是一类很特殊的函数，它能返回一个值。我们在之前已经遇到过。在练习3-7中我们使用abs() 函数来计算一个数字的绝对值。一个函数能够计算出一个值并且返回，我们将在第7章中充分探讨这个，但是我们将需要一些时间来介绍一下现在的想法。

不像我们看到的大多数函数（如line(), ellipse(), 以及 rect() 等等）。random() 不需要在屏幕上绘制或者填充什么颜色。相反，random() 能够回答这问题，它会返回一个值给我们。这里有一段对话，你可以随时跟你的朋友练练。

我：嘿，随机，最近怎么样？听着我只想知道你能不能给我一个在1到100之间的一个随机数？

随机：很好，没问题。63怎么样？

我：很好！谢谢！我就要画一个63像素宽的矩形。

现在，让我们来看看在Processing环境中正式的用语应该是怎样？如下代码中的“我”就是“w”。

```
float x = random(1,100);    ← 一个1到100中随机的浮点数（小数）。
rect(100,100,w,50);
```

函数 random() 需要2个参数，并且它会返回在那2个参数中的一个随机浮点数。第2个参数必须大于第一个参数，它才能够正常工作。函数 random() 当然也可以设置一个参数，那么它的会默认随机值在0到你设置的那个参数之间。

此外，random() 只会返回浮点数。这就是为什么我们在声明 “w” 的时候是声明的浮点数。如果你想要随机一个整数，你也可以将结果转换为整数形式的随机值。

```
int w = int(random(1,100));    ← 一个1到100中随机的整数。
rect(100,100,w,50);
```

注意使用嵌套的括号。这时一个很好的习惯，很方便我们在函数内部调用函数。random() 函数返回一个浮点值，然后它通过int()函数转换成一个整数。如果我们想用简单的嵌套功能，我们可以直接写成一行：

```
rect(100,100,int(random(1,100)),50);
```

顺便说一下，从一个数据类型转换到另一个数据类型的过程被称为“casting.”（铸造），在Java（Processing基于Java）中将浮点数铸造成整数也可以用这种方式编写：

```
int w = (int) random(1,100);    ← 结果是1到100的中一个随机的浮点数。它通过“casting”转换成整数。
```

OK,我们现在对random() 进行试验。例4-7显示了如果我们把每个随机变量纺织在圆圈（填充颜色，位置，大小）中会发生什么，并且每个循环后将其分配到不一样的随机数。如图4.8所示。

例 4-7：在变量中使用随机值

```

float r;
float g;
float b;
float a;

float diam;
float x;
float y;
void setup() {
    size(200,200);
    background(0);
    smooth();
}

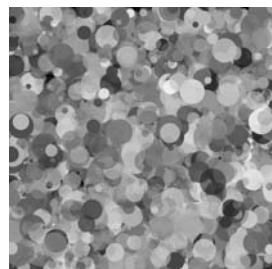
void draw() {
    // Fill all variables with random values
    r = random(255);
    g = random(255);
    b = random(255);
    a = random(255);
    diam = random(20);
    x = random(width);
    y = random(height);

    // Use values to draw an ellipse
    noStroke();
    fill(r,g,b,a);
    ellipse(x,y,diam,diam);
}

```

每一次循环draw()，就会重新随机一次它们的值。

图 4.8



4.7 变量 Zoog

我们现在已经准备好重做Zoog了，我们的外星人朋友，当我们完成之后，他会跟随鼠标在屏幕周围到处跑。在这里，我们添加2个特点给Zoog。

- **新特点 #1** — Zoog将从屏幕下放飞到屏幕上方。
- **新特点 #2** — 当Zoog在移动时，它的眼睛颜色会随机变化。

新特点 #1能够使用我们先前所学的mouseX和mouseY，以及带入我们的变量轻松解决。

新特点 #2 通过创建另外3个变量eyeRed, eyeGreen, 和 eyeBlue 来实现，他将用于 fill()函数，显示之前的眼睛。

例 4-8：变量 Zoog

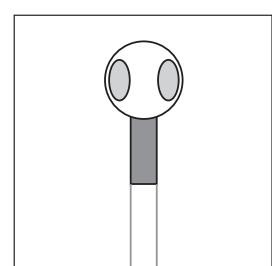
```

float zoogX;
float zoogY;
float eyeR;
float eyeG;
float eyeB;
void setup() {
    size(200,200);

    // 声明函数。zoogX, zoogY用于特点 #1。
    // eyeR, eyeG, eyeB用于特点 #2。
}

```

图 4.9



```
zoogX = width/2;           // Zoog always starts in the middle
zoogY = height + 100;      // Zoog starts below the screen
smooth();
}
```

```
void draw() {
    background(255);

    // Set ellipses and rects to CENTER mode
    ellipseMode(CENTER);
    rectMode(CENTER);
```

```
// Draw Zoog's body
stroke(0);
fill(150);
rect(zoogX,zoogY,20,100);
```

```
// Draw Zoog's head
stroke(0);
fill(255);
ellipse(zoogX,zoogY - 30,60,60);
```

```
// Draw Zoog's eyes
eyeR = random(255);
eyeG = random(255);
eyeB = random(255);
fill(eyeR,eyeG,eyeB);
ellipse(zoogX - 19,zoogY - 30,16,32);
ellipse(zoogX + 19,zoogY - 30,16,32);
```

```
// Draw Zoog's legs
stroke(150);
line(zoogX - 10,zoogY + 50,zoogX - 10,height);
line(zoogX + 10,zoogY + 50,zoogX + 10,height);
```

```
} // Zoog moves up
zoogY = zoogY - 1;
```

特点 #1的zoogX和zoogY的初始化时基于窗口的尺寸。请注意我们在函数被调用之前不能初始化变量，因为我们使用的是内置变量width和height。

特点 #1的zoogX和zoogY用于形状的位置。

特点 #2中eyeR, eyeG, eye将给予随机值，并且用于fill()函数中。

练习4-6：修改例 4-8，让 Zoog 从进行左右移动。提示：这需要 random() 结合在 zoogX 中使用。



zoogX = _____;



练习4-7：使用变量以及随机函数，修改你第一课的项目，让他们在屏幕上动起来，并改变颜色，尺寸，位置等等。

5. 条件

“语言是雷人理性的工具，不是表达思想的媒介，这是一个普遍存在的真理”

— 乔治·布尔 (George Boole)

本章内容：

- 布尔表达式
- 条件语句：根据不同的情况产生不同的结果
- If, Else If, Else

5.1 布尔表达式

什么类型的测试是你最喜欢的？简答题？选择题？在计算机的编程世界里，我们只需要一个类型的测试题：布尔测试 — 真或假。布尔表达式（数学家乔治·布尔命名）是表达真还是假的表达式。让我们来看看一些常见的语言例子：

- 我饿了 → 真
- 我害怕电脑编程 → 假
- 这本书很搞笑 → 假

在计算机的科学逻辑中，我们可以表示数字之间的关系。

- 15大于20 → 假
- 5等于5 → 真
- 32小于等于33 → 真

在本章中，我们将学习如何在布尔表达式中使用变量，通过我们的草图根据现在的值存储变量。

- $x > 20$ → 取决于现在的x值
- $y == 5$ → 取决于现在的y值
- $z < 33$ → 取决于现在的z值

下面的运算符是一些我们能够使用的布尔表达式。

>	大于	<=	小于等于
<	小于	==	等于
>=	大于等于	!=	不等于

5.2 条件语句：If, Else, ElseIf

布尔表达式（通常被称为“条件”）的操作会在草图中产生问题。15大于20？如果这个答案是yes（就是真），我们就能执行一些指令（如画一个矩形）；如果这个答案是no（就是假），这些要执行的指令就忽略掉。这个分支想法的引入取决于各种条件，程序可以按照不同的条件运行。

在物理世界中，可能指令就像这样：

如果我饿了我就要找一些食物吃，否则如果我渴了我就要找一些水喝，否则就小睡一会儿。

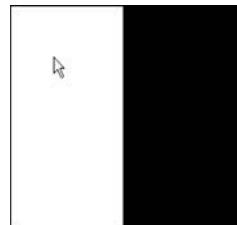
在 Processing 中，指令可能更像这些：

如果鼠标在屏幕的左侧，就在左侧绘制一个矩形。

或者更正式的说，可能就像图5.1里面的一样

```
if (mouseX < width/2) {
    fill(255);
    rect(0,0,width/2,height);
}
```

图 5.1



上面的布尔表达式以及所得的指令被放在一个代码块中，语法和结构如下：

```
if (boolean expression) {
    // code to execute if boolean expression is true
}
```

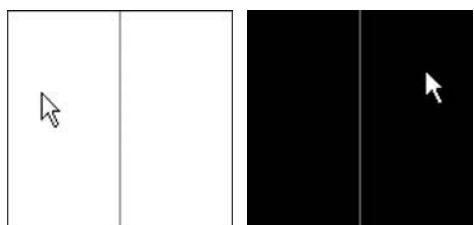
如果布尔表达式为假，这个结构可以结合其他的关键词进行扩展。这就相当于“否则，就这样做”。

```
if (boolean expression) {
    // code to execute if boolean expression is true
} else {
    // code to execute if boolean expression is false
}
```

如果鼠标在屏幕左侧，绘制出白色背景，否则，绘制一个黑色背景。

图 5.2

```
if (mouseX < width/2) {
    background(255);
} else {
    background(0);
}
```



最后我们要用多个条件我们能够使用“else if。”当使用一个else if时，条件语句将进行评估给出顺序。只要找到一个布尔表达式时真的，才会执行相应代码，剩下的布尔表达式会被忽略。如图5.3所示。

```
if (boolean expression #1) {
    // code to execute if boolean expression #1 is true
} else if (boolean expression #2) {
    // code to execute if boolean expression #2 is true
} else if (boolean expression #n) {
    // code to execute if boolean expression #n is true
} else {
    // code to execute if none of the above
    // boolean expressions are true
}
```

我们举一个鼠标的例子进一步了解，我们可以说下面的else，结果如图5.4所示。

如果鼠标在最3个栏中的最左边的栏中，那么就绘制一个白色的背景，如果鼠标在中间的栏中，那么就绘制一个灰色背景，否则，鼠标就绘制一个黑色背景。

```
if (mouseX < width/3) {
    background(255);
} else if (mouseX < 2*width/3) {
    background(127);
} else {
    background(0);
}
```

图 5.4

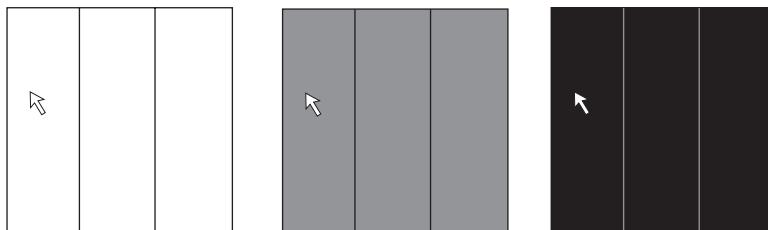
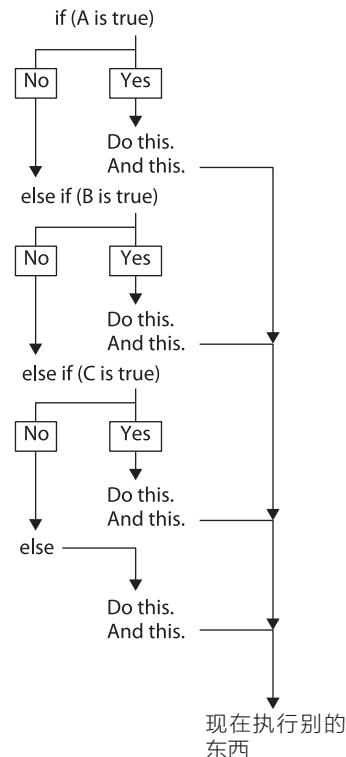


图 5.3



练习5-1：思考一个数字分级系统（ABCD这类似的评分等级）。填写下面空白的代码完成你的布尔表达式。

```
float grade = random(0,100);
if (_____) {
    println("Assign letter grade A. ");
} else if (_____) {
    println(_____);
} else if (_____) {
    println(_____);
}
```

在传统的表达式中你只能有一个if，一个else。但是你可以有多少个else if。

```

} else if (_____){
    println(_____);
} else {
    println(_____);
}

```

练习5-2：检查下面的代码示例，并确定在消息窗口会弹出什么。写下你确定的答案，然后执行一次看是否和你写得答案一样。



问题1：下面哪个例子能够分开一个数是在0到25之间，26到50之间或者大于50这几个等级？

```
int x = 75;
```

```

if (x > 50) {
    println(x + " is greater than 50! ");
} else if (x > 25) {
    println(x + " is greater than 25! ");
} else {
    println(x + " is 25 or less! ");
}

```

输出结果:_____

```
int x = 75;
```

```

if(x > 25) {
    println(x + " is greater than 25! ");
} else if (x > 50) {
    println(x + " is greater than 50! ");
} else {
    println(x + " is 25 or less! ");
}

```

输出结果:_____

问题2：下面哪个例子输出为当一个数是5的时候将它变成6,当一个数是6的时候将它变成5?

```
int x = 5;
```

```

println("x is now: " + x);
if (x == 5) {
    x = 6;
}
if (x == 6) {
    x = 5;
}
println("x is now: " + x);

```

输出结果:_____

```
int x = 5;
```

```

println("x is now: " + x);
if (x == 5) {
    x = 6;
} else if (x == 6) {
    x = 5;
}
println("x is now: " + x);

```

输出结果:_____

需要指出的是在练习5-2中，如果我们测试2个值相等我们需要用2个等号。因为在编程是1个等号代表的是赋值。

if (x == y) {

当要问x是否等于y时用2个等号

x = y;

当要说将x赋予y值时用1个等号

5.3 在草图中使用条件语句

让我们来看一个非常简单的例子，在一定条件下执行不同的任务。这个代码如例5.1所示。

步骤1 创建变量来保存红、绿、蓝颜色的份量。我们声明叫作r,g,b。

步骤2 根据这些颜色不断绘制背景。

步骤3 如果鼠标在屏幕的右侧，r的值会递增，如果鼠标在屏幕的右侧，r的值会递减。

步骤4 让r的值在0到255之间

例 5-1：条件语句

```
float r = 150;
float g = 0;
float b = 0;
```

```
void setup() {
    size(200,200);
}
```

```
void draw() {
    background(r,g,b);
    stroke(255);
    line(width/2,0,width/2,height);
```

```
if(mouseX > width/2) {
    r = r + 1;
} else {
    r = r - 1;
}
```

```
if (r > 255) {
    r = 255;
} else if (r < 0) {
    r = 0;
}
```

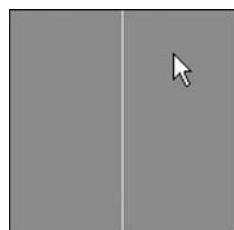
1. 变量申明初始化

2. 画东西

3. “如果鼠标在屏幕右侧”等于“如果mouseX大于屏幕宽度的一半”

4. 如果r大于255,就让它等于255;
如果r小于0, 就让它等于0

图 5.5



在第4步中限制变量的值是一个常见的问题。这里，我们不希望颜色值增加或减少到不合理的极端值。在其他例子中，我们可能要约束形状，不会让它过大或过小。

虽然使用if语句是一个很有效果的限制方式，但是Processing也提供了一个限制的函数 constrain()，它能够用一行代码取代这个if语句。

```
if (r > 255) {
    r = 255;
} else if (r < 0) {
    r = 0;
}
r = constrain(r,0,255);
```

用if语句进行限制

用限制函数进行限制

constrain() 函数需要三个参数：我们需要限制的值，最低限制，最高限制。函数会返回“限制”的值，并且分配给变量r。（请记住函数返回值是什么意思，我们在random()中讨论过。）

将限制值当作编程习惯能够很好的避免一些错误的发生；无论怎样你都能确保你的变量在一个给定的范围内变动。有一天，当你在做大型软件项目并与多个程序员一起工作时，constrain() 函数能够很好的确保它的那部分代码很好的工作。

让我们把我们的第一个例子改进的更好一些并且根据鼠标移动的位置和点击状态能同时更改3个颜色值的份量。注意constrain() 需要用到的3个值。mousePressed系统变量是真还是假取决于用户是否按住了鼠标。

例 5-2：更多条件语句

图 5.6

```
float r = 0;
float b = 0;
float g = 0;
```

关于背景颜色的3个变量

```
void setup() {
    size(200,200);
}
```

```
void draw() {
    background(r,g,b);
    stroke(0);
```

给背景设置颜色，画线把窗口分成4个象限。

```
line(width/2,0,width/2,height);
line(0,height/2,width,height/2);
```

```
if(mouseX > width/2) {
    r = r + 1;
} else {
    r = r - 1;
}
```

如果鼠标在窗口右侧红色会递增；反之，鼠标在窗口左侧红色会递减。

```
if (mouseY > height/2) {
    b = b + 1;
} else {
    b = b - 1;
}
```

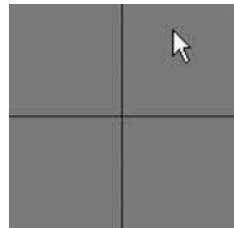
如果鼠标在窗口下半部分蓝色会递增；反之，鼠标在窗口上半部分蓝色会递减。

```
if (mousePressed) {
    g = g + 1;
} else {
    g = g - 1;
}
```

如果鼠标按下绿色会递增（使用系统变量mousePressed）。

```
r = constrain(r,0,255);
g = constrain(g,0,255);
b = constrain(b,0,255);
}
```

限制r, g, b的取值范围从0到255。



练习5-3：通过变量递增在一个窗口中移动矩形。矩形x坐标从0开始，使用if语句让它在100的时候停下来。重写草图使用constrain()函数代替if语句。填写如下缺少的代码。



```
// Rectangle starts at location x
float x = 0;

void setup() {
    size(200,200);
}

void draw() {
    background(255);

    // Display object
    fill(0);

    rect(x,100,20,20);
    // Increment x
    x = x + 1;
    _____
    _____
    _____
}
```

5.4 逻辑运算符

我们已经征服了简单的if语句

如果我的体温大于98.6度，请带我去看医生。

然而有时候简单地执行一个条件是不够的。例如：

如果我的体温大于98.6度或者我的受伤长皮疹了，请带我去看医生。

如果我被蜜蜂蛰了并且我对蜜蜂过敏，请带我去看医生。

我们通常会希望在编程的时候时不时做一些同样的事。

当鼠标在屏幕右侧以及在屏幕底部的时候，在矩形右下角画出一个矩形。

你的第一反应可能事在上面的代码中嵌套一个if语句，就像这样：

```
if (mouseX > width/2) {
    if (mouseY > height/2) {
        fill(255);
        rect(width/2,height/2,width/2,height/2);
    }
}
```

换句话说，我们不得不通过2个if语句来执行那些我们想要的。这是可行的，但是我们有一个更简单的方法来完成，我们称之为“并且符号”，2个就像这样“`&&`”。一个单一的逻辑之和符号表示你可能在else1中有什么，所以一定要写2个。

`|| (或)`
`&& (并且)`
`! (不是)`

“或者”是2条竖线（又名“管道”）“`||`”。如果你在键盘上找不到它的位置，它通常在斜杠键的上面。

```
if (mouseX > width/2) {
    if (mouseY > height/2) {
        fill(255);
        rect(width/2,height/2,width/2,height/2);
    }
}
```

此外不仅仅有`&&`和`||`，我们有时也会运用到逻辑运算符“不是”，写成一个惊叹号：`!`。

我的体温不高于98.6，我不会打电话请病假的。
 如果我被蜜蜂蛰了但我并没有对蜜蜂过敏，不用担心！

下面是一个Processing的例子：

如果鼠标没有按下，请绘制一个圆，否则就绘制一个正方形。

```
if (!mousePressed) {
    ellipse(width/2,height/2,100,100);
} else {
    rect(width/2,height/2,100,100);
}
```

! 表示没有，“`mousePressed`”是作为布尔表达式的一个布尔变量。它的值是真或假（取决于当前是否按下了鼠标）。布尔变量我们将在5.6节中详细探讨。

请注意也可以不需要“不是”编写：

如果鼠标按下，请绘制一个正方形，否则就绘制一个圆形。



练习5-4：下面的布尔表达式是真或者假？假设变量x=5, y=6。

!(x > 6) _____
 (x==6 && x==5) _____
 (x==6 || x==5) _____
 (x>1 && y<10) _____

Although the syntax is correct, what is flawed about the following boolean expression?

(x > 10 && x < 5) _____



练习5-5：编写一个简单翻转的编程。换句话说，如果鼠标在矩形上，矩形就会改变颜色。下面是一些代码，作为你的开始。

```
int x = 50;
int y = 50;
int w = 100;
int h = 75;

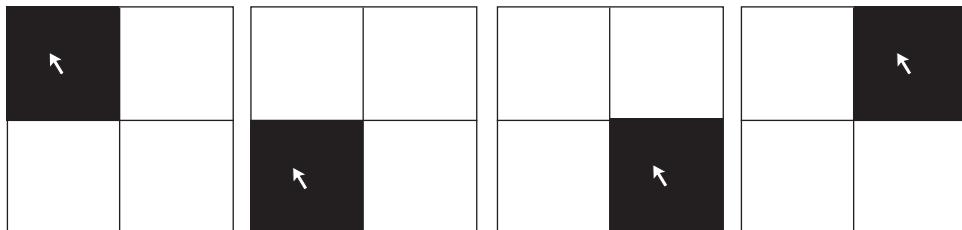
void setup() {
  size(200,200);
}

void draw() {
  background(0);
  stroke(255);
  if (_____ && _____ && _____ && _____) {
    _____
  } _____ {
    _____
  }
  rect(x,y,w,h);
}
```

5.5 多个图像变化

让我们用一个稍微简单的方法解决这些问题。思考以下在一个单独的草图中实现如图5.7的样子。黑色方形会显示在鼠标当前的象限中。

图 5.7



首先让我们先写处代码的逻辑（用英语）。

Setup:

- 1.设置一个200*200px的窗口

Draw:

- 1.绘制一个白色背景
- 2.绘制一个水平线和垂直线将窗口分成四个象限
- 3.如果鼠标在左上角象限，绘制一个黑色在左上角象限
4. 如果鼠标在右上角象限，绘制一个黑色在右上角象限
5. 如果鼠标在左下角象限，绘制一个黑色在左下角象限
6. 如果鼠标在右下角象限，绘制一个黑色在右下角象限

对于3~6点的步骤，我们必须要问自己一个问题：“我们怎么知道鼠标在给定的象限中？”需要做到这一点我觉需要制定一个更具体的if语句。比如我们会说：“如果鼠标的X坐标大于100px并且鼠标Y坐标大于100px，在右下角绘制一个黑色矩形。作为一个练习，你需要根据上面的伪代码写自己的代码。答案仅供参考，如例5-3所示。

例 5-2：更多条件语句

```
void setup() {
    size(200,200);
}

void draw() {
    background(255);
    stroke(0);
    line(100,0,100,200);
    line(0,100,200,100);
```

```
// Fill a black color
noStroke();
fill(0);

if (mouseX < 100 && mouseY < 100) {
  rect(0,0,100,100);
} else if (mouseX > 100 && mouseY < 100) {
  rect(100,0,100,100);
} else if (mouseX < 100 && mouseY > 100) {
  rect(0,100,100,100);
} else if (mouseX > 100 && mouseY > 100) {
  rect(100,100,100,100);
}
}
```

每个象限的不同显示，取决于鼠标当前所在的位置。



练习 5-6：重新编写例5-3的代码，让它当鼠标离开那个象限时从黑色过渡到白色。注意“你需要四个变量，每一个变量针对每一个象限。

5.6 布尔变量

毕竟，按钮在点击时只有一个响应。现在，你可能会对图像变化以及按钮有一些失望。也许你会像：“我不能只从菜单或者什么地方选择‘添加按钮’吗？”对于我们来说，现在的答案是不能。在最终我们会学会如何使用从代码库使用代码（可能你会使用代码库来创建按钮使其更容易），但是需要从头学习如何编程GUI（图形用户界面）。

第一点，练习按钮，变化编程是一个学习条件和变量的很好的学习方式。第二，使用老套相同的按钮编程和变化是很无聊的。如果你对新的交互界面很关心和有兴趣，你练习这些就是你所需要的基本技能。

好的，现在我们来看看如何使用一个布尔变量编程一个按钮操作。一个布尔变量（或者说是一个布尔类型的变量）它的答案只有真或假。把它想象成一个开关。它只有开或关。按下按钮，开关打开；再次按下按钮，开关关闭。我们只需要例5-2中的布尔变量：内置变量`mousePressed`。当鼠标按下的时候`mousePressed`为真，当鼠标没有按下的时候`mousePressed`为假。

所以我们布尔变量的例子将包括一个初始值为假的布尔变量（假设是在关机状态下开始的）。

```
boolean button = false;
```

一个布尔变量只包含真或假。

在鼠标指向效果时，任何时候鼠标停在矩形上它就会变成白色。我们的草图会在鼠标按下的时候变成黑色。

```
if (button) {
  background(255);
} else {
  background(0);
}
```

如果`button`值是真的，背景就是白色，

否则都是黑色

我们能够检查以下，如果鼠标的位置在矩形内，如果鼠标按下按钮，设定真或假出来的相应结果是怎样。下面是一个完整的例子：

例 5-4：按下按钮

```
boolean button = false;
int x = 50;
int y = 50;
int w = 100;
int h = 75;

void setup() {
    size(200,200);
}
void draw() {
    if (mouseX > x && mouseX < x + w && mouseY > y && mouseY < y + h && mousePressed) {
        button = true;
    } else {
        button = false;
    }

    if (button) {
        background(255);
        stroke(0);
    } else {
        background(0);
        stroke(255);
    }

    fill(175);
    rect(x,y,w,h);
}
```

如果鼠标在矩形中 (mouseX, mouseY)，并且
鼠标按下了那么就是真。

这个示例模拟了一个连接到灯的按钮动作，只要你按下了就会亮，如果你放手，灯就会熄灭。或许它在某些互动中是一个很好的表现形式，但这不是我们本节要学的。我们需要学的是一个开关，如果指示灯不亮我们就让它开开，如果是亮的我们就关闭它。

要让它正常运行，我们必须检查并确定绘制矩形的代码是否在mousePressed() 里面而不是在draw() 里面。根据定义，当用户点击鼠标，mousePressed()中的代码会执行并且只执行一次（参考3.4节）。当鼠标点击，我们就会让开关开启或关闭（一次且仅一次）。

我们现在需要编写一些代码来“切换”开关，改变他们从开到关，从关到开的状态。这个代码会放在mousePressed()中。

如果变量“button”为真，那么我们应该把它变成假。如果是假我们就把它变成真。

```
if (button) {
    button = false;
} else {
    button = true;
}
```

用一个明确的方式切换布尔值。如果按钮是
真，它会变成假。否则，按钮是假，它会变成
真。

这里右一个简单的方法实现这个功能：

```
button = !button;
```

真会变成假，假会变成真。

在这里，所需要的值就是这个这个值的“否”值。换句话说，如果按钮是真，它会变成假，如果按钮是假，它就会变成真。有了这个奇怪但有效的代码，我们再看看例5-5中的按钮示例。

例 5-5：开关按钮

```
boolean button=false;
```

```
int x= 50;
int y=50;
int w=100;
int h=75;

void setup(){
    size(200,200);
}

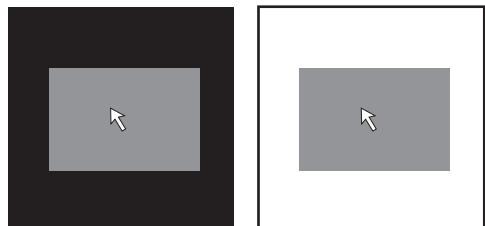

```

```
void draw(){
    if(button){
        background(255);
        stroke(0);
    } else{
        background(0);
        stroke(0);
    }

    fill(175);
    rect(x,y,w,h);
}
```

```
void mousePressed(){
    if(mouseX>x && mouseX<x+w && mouseY>y && mouseY<y+h ){
        button= !button;
    }
}
```

图 5.8



当鼠标按下时，按钮状态就开始反向运作。尝试将代码移动到draw() 中试一试。（参见 练习5 - 7）。

练习5-7：为什么在将下列代码放入 draw() 的时候，它就不正常工作了？



```
if(mouseX>x && mouseX<x+w && mouseY>y && mouseY<y+h ){
    button= !button;
}
```

练习5-8：在之前的章节例4-3中一个圆从窗口移动。修改草图内容，让它使鼠标一旦被按下才开始移动。



```
boolean _____ = _____;

int circleX = 0;
int circleY = 100;

void setup() {
    size(200,200);
}
void draw() {
    background(100);
    stroke(255);
    fill(0);
    ellipse(circleX,circleY,50,50);

}

void mousePressed() {
}
```

5.7 弹弹球

现在是时候让我们再次回到我们的朋友Zoog上。让我们回顾一下到目前为止我们已经做了什么。首先，我们学会了从Processing参考中利用形状函数绘制Zoog。之后，我们意识到可以使用变量代替直接编码。这些变量代码允许我们移动Zoog。如果Zoog的位置在X,从X开始画它，接着是X+1,X+2，以此类推。

这是令人兴奋但也悲伤的。我们经历了从发现到运动的乐趣，我们很快就要看到Zoog离开屏幕了。幸运的是，条件语句在这里有所保留，请允许我们问一个问题：如果Zoog已经到达了屏幕边缘，咱们把边缘给封闭，让它留在屏幕好不好？

为了让事情简化，我们首先从一个简单的圆开始。

写一个简单的编程，Zoog（用圆来代替），让它从屏幕的左边移到屏幕的右边，当到达右边边缘的时候让它调转方向。

从先前的章节中我们知道我们需要一个全局变量来跟踪Zoog的位置。

```
int x = 0;
```

这就足够了吗？并不是。在我们先前的例子中Zoog要始终保持1像素的移动。

```
x = x + 1;
```

这是告诉Zoog让它往右移动。但是如果我们希望它往左移动呢？这很简单端对不对？

```
x = x - 1;
```

换句话说，有时候Zoog的移动速度是“+1”，有时候是“-1”。它的运动速度是一个变量。是的，铃儿当当响。为了切换Zoog速度的方向，我们需要另外一个变量：speed。

```
int x = 0;  
int speed = 1;
```

为Zoog的速度设置一个变量。当Zoog向右边移动时，速度为正；当Zoog向左边移动时，速度为负。

现在我们有了我们的变量，接下来我们看看剩下的代码。假设我们已经设置了窗口的尺寸，我们可以直接看看draw()中所需的步骤。我们能够把Zoog看作一个球，因为现在我们只画一个圆。

```
background(0);  
stroke(255);  
fill(100);  
ellipse(x,100,32,32);
```

为了简化，Zoog在这里就是一个圆。

这些都是初级的东西。现在为了让球动起来，它的x坐标值应该随着draw()重复。

```
x = x + speed;
```

如果我们现在运行程序，圆会从窗口左侧开始移动到右侧，并且越过屏幕继续向右—这些结果我们在第4章就已经完成了。为了让它转向，我们需要一个条件语句。

如果球到了屏幕边缘，让它转向。

或者，更正式一点的说：

如果x大于width，反向运动。

```
if (x > width) {  
    speed = speed * -1;  
}
```

乘以-1变成反向速度。

反向转化数字的正负值

当我们需要反向转化数字的值时，我的意思是希望让一个正数变成负数时，或者负数变成正数时。可以时通过乘以或除以-1来获得。下面的公式是一个小提醒。

- $-5 * -1 = 5$
- $-5 * -1 = -5$
- $-1 * 1 = -1$
- $-1 * -1 = 1$

现在运行草图，我们能够看到圆在运动到窗口右边缘时开始反向运动到屏幕左边缘，但是它会穿过屏幕左边缘。这里我们需要稍稍修改一下条件。

如果球到达右边缘或者左边缘，让它转向。

或者，更正式一点的说：

如果x大于width或者x小于0时，反向运动。

```
if ((x > width) || (x < 0)) {  
    speed = speed * -1;  
}
```

请记住 || 的意思是“或者”

例5-6把他们放在一起。

例 5-6： 弹弹球

```
int x = 0;  
int speed = 1;  
  
void setup() {  
    size(200,200);  
    smooth();  
}  
  
void draw() {  
    background(255);  
    x = x + speed;  
  
    if ((x > width) || (x < 0)) {  
        speed = speed * -1;  
    }  
  
    // Display circle at x location  
    stroke(0);  
    fill(175);  
    ellipse(x,100,32,32);  
}
```

在x中添加一个当前的速度

如果x到达边缘，速度乘以-1，让它开始反向运行。



练习5-9：重写例5-6，让它不仅仅能够在水平方向反向运动，在垂直方向也能。你能在它的基础上实现其他的功能吗，如基于一定条件下改变圆的尺寸或者颜色？除了改变方向，你可以使球的速度加快或者减慢吗？

例 5-7：“弹弹”颜色

```

float c1 = 0;
float c2 = 255;
float c1dir = 0.1;
float c2dir = -0.1;

void setup() {
    size(200,200);
}

void draw() {
    noStroke();

    // Draw rectangle on left
    fill(c1,0,c2);
    rect(0,0,100,200);

    // Draw rectangle on right
    fill(c2,0,c1);
    rect(100,0,100,200);

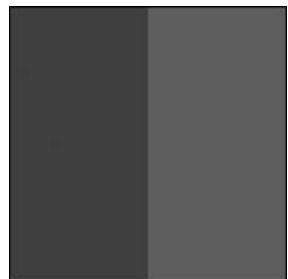
    // Adjust color values
    c1 = c1 + c1dir;
    c2 = c2 + c2dir;

    // Reverse direction of color change
    if (c1 < 0 || c1 > 255) {
        c1dir *= -1;
    }

    if (c2 < 0 || c2 > 255) {
        c2dir *= -1;
    }
}

```

图 5.9



这里取代了之前的窗口边缘反向，而是当色彩值小于0或大于255这两个边缘的时候反向。这就像弹弹球。

条件语句在我们的变成工具中允许更复杂的运动。例如，思考一下举行跟在窗口的边缘。

解决方式之一就是想想在矩形边缘运动的4个状态，这里从状态0到状态3。如图5.10所示。

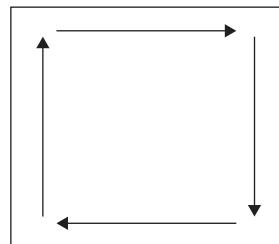
- 状态#0：从左到右
- 状态#1：从上到下
- 状态#2：从右到左
- 状态#3：从下到上

我们可以利用一个变量来跟踪这些状态数，并且按照状态的不同调整它的x, y坐标值。例如：如果在状态#2,x等于x减1。

一旦矩形移动到了那个状态的顶点，我们能够改变它的状态变量。如果是在状态#2：（1）x等于x减1（2）如果x小于零，跳到状态3。

下面的例子实现了这个复杂的逻辑。

图 5.10



例 5-8：矩形在窗口边缘运动，运用“状态”变量

```
int x = 0; // x location of square
int y = 0; // y location of square

int speed = 5; // speed of square

int state = 0;

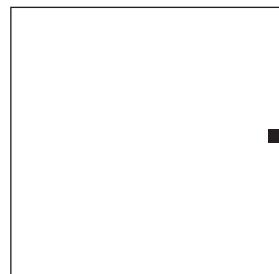
void setup() {
    size(200,200);
}

void draw() {
    background(100);

    // Display the square
    noStroke();
    fill(255);
    rect(x,y,10,10);

    if (state == 0) {
        x = x + speed;
        if (x > width-10) {
            x = width-10;
            state = 1;
        }
    } else if (state == 1) {
        y = y + speed;
        if (y > height-10) {
            y = height-10;
            state = 2;
        }
    } else if (state == 2) {
        x = x - speed;
    }
}
```

图 5.11



如果为状态 # 0，向右移动。

当它的状态为0，并且到达了窗口右侧时，改变它的状态为1。在所有状态下重复这个相同的逻辑。

```

if (x < 0) {
  x = 0;
  state = 3;
}
} else if (state == 3) {
  y = y - speed;
  if (y < 0) {
    y = 0;
    state = 0;
  }
}
}

```

5.6 弹弹球

对于我来说，我的编程生活中最快乐的时刻就是我意识到我可以编写重力的时候。实际上，你知道了变量和条件，你就已经做好了准备。

弹弹球的草图告诉我们，一个无题通过速度移动来改变它的位置。

```
location = location + speed
```

重力是一种强制性的动力。当你放下港币，它就会随着地球的重力加速到地面上。所以我们必须增加我们弹弹球的“加速度”（这是由重力引起的，但是可以是不一样的数值）这个概念。加速度的增加（或者减少）。换句话说，加速度是速度变化的速率，速度是变化速率的初始位置。我们只需要另一行代码：

```
speed = speed + acceleration
```

现在，让我们来一个简单的重力模拟。

例 5-9：简单重力模拟

```
float x = 100; // x location of square
float y = 0; // y location of square
```

```
float speed = 0; // speed of square
float gravity = 0.1;
```

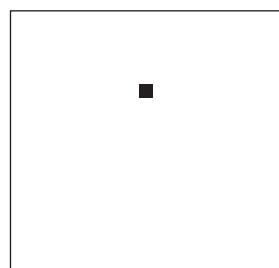
```
void setup() {
  size(200,200);
}
```

```
void draw() {
  background(255);
```

```
// Display the square
fill(0);
noStroke();
rectMode(CENTER);
```

一个新的变量，重力（即加速度）。我们使用的数量相对较少（0.1）因为这个数字会随着时间的积累越来越快。试一试把这个数字改成2.0，看看会发生什么。

图 5.12



```

rect(x,y,10 , 10);

y = y + speed;           添加速度。

speed = speed + gravity; 添加加速度。

// If square reaches the bottom
// Reverse speed
if (y > height) {
    speed = speed * -0.95;   乘以-0.95而不是-1是为了减缓每次反弹的距离（下降速度）。这就是所谓的“阻尼”效果，它更多的是模拟真实世界的重力（没有它，这个球会永远反弹）。
}
}

```

练习 5-10：继续你的设计，并且在本章中添加一些演示功能。一些方向：

- 当鼠标经过的时候，让你的设计改变某些颜色。
- 让矩形在屏幕中四处移动，你能让其窗口边缘全部关闭，在窗口中来回弹吗？
- 颜色的淡入淡出。

这是一个有Zoog的简单版本。

例 5-10：Zoog 和条件

```

float x = 100;
float y = 100;
float w = 60;
float h = 60;
float eyeSize = 16;

```

```

float xspeed = 3;
float yspeed = 1;

```

Zoog在水平方向和垂直方向有一个速度的变量。

```

void setup() {
    size(200,200);
    smooth();
}

void draw() {
    // Change the location of Zoog by speed
    x = x + xspeed;
    y = y + yspeed;
}

```

100

```
if ((x > width) (x < 0)) {  
    xspeed = xspeed * -1;  
}  
  
if ((y > height) (y < 0)) {  
    yspeed = yspeed * -1;  
}  
  
background(0);  
ellipseMode(CENTER);  
rectMode(CENTER);  
noStroke();  
  
// Draw Zoog's body  
fill(150);  
rect(x,y,w/6,h*2);  
  
// Draw Zoog's head  
fill(255);  
ellipse(x,y-h/2,w,h);  
  
// Draw Zoog's eyes  
fill(0);  
ellipse(x-w/3,y-h/2,eyeSize,eyeSize*2);  
ellipse(x + w/3,y-h/2,eyeSize,eyeSize*2);  
  
// Draw Zoog's legs  
stroke(150);  
line(x-w/12,y + h,x-w/4,y + h + 10);  
line(x + w/12,y + h,x + w/4,y + h + 10);  
}
```

一个带有“或者”的if语句，确定Zoog是否到达了窗口左边缘或者右边缘。如果到了，我们就让速度乘以-1，让Zoog反向移动。

相同的逻辑应用在y方向上。

6. 循环

“重复对于生活来说是现实的以及严重的”

— Soren Kierkegaard

本章内容：

- 迭代的概念
- 两种类型的循环：“while” 和 “for”，以及什么时候用它们
- 对计算机图形的内容迭代

6.1 什么是迭代？我的意思是，什么是迭代？说真的，什么是迭代？

迭代就是一遍又一遍重复同一种规则或步骤的过程。它是计算机编程的基本概念，我们很快就会发现它会使我们的编程变得更佳有趣。让我们开始吧。

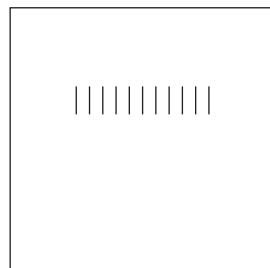
现在，让我们想想腿。很多很多的腿腿在我们Zoog身上。如果我们只看过这本书的第1章，我们可能会写一些如例6-1的代码。

例 6-1：一些线

```
size(200,200);
background(255);

// Legs
stroke(0);
line(50,60,50,80);
line(60,60,60,80);
line(70,60,70,80);
line(80,60,80,80);
line(90,60,90,80);
line(100,60,100,80);
line(110,60,110,80);
line(120,60,120,80);
line(130,60,130,80);
line(140,60,140,80);
line(150,60,150,80);
```

图 6.1



在上述示例中，腿从x = 50一直到x = 150px，每隔10px会有一个腿。当然这些代码可以实现这些，但是，我们在第4章学习了变量，我们可以作出一些实质性的改善，并且消除这些硬编码值。

首先，我们需要设置我们系统的参数变量：腿的x, y值，长度，以及腿与腿只见的距离。注意在绘制腿的时候，只有x的值在变化。所有其他变量保持不变（但他们可能随时会改变，如果我们系w那个他们改变）。

例 6-1：一些线

```

size(200,200);
background(0);

// Legs
stroke(255);

int y = 80; // Vertical location of each line
int x = 50; // Initial horizontal location for first line
int spacing = 10; // How far apart is each line
int len = 20; // Length of each line

line(x,y,x,y + len); 画第一条腿。

x = x + spacing;
line(x,y,x,y + len); 在与下一个腿之间添加  
10px的距离

x = x + spacing;
line(x,y,x,y + len);

x = x + spacing;
line(x,y,x,y + len); 继续重复以上的过程。

x = x + spacing;
line(x,y,x,y + len);

```

不是太糟糕，我想。奇怪的是尽管在技术上更先进（我们可以调整间距的变量，只需要改变一行代码），但是我们还是落后一步，产生了近2倍的代码之多！如果我们想绘制100条腿呢？对于每一条腿我们需要两行代码吗。那么这100条腿就需要200行代码！为了避免这种可怕的结果，我们会希望这样说：

绘制出一条线一百次。

哈哈，只用了一行代码！

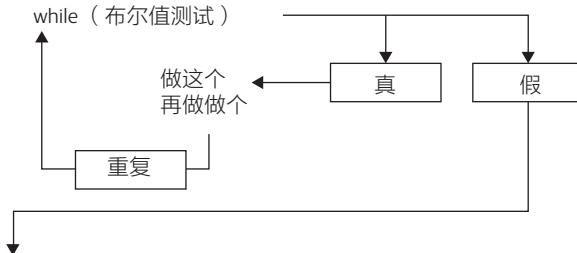
显然我们不是第一个遇到这种困境的程序员，并且这可以使用常用的方法解决—循环。一个循环结构和条件语句的语法类似（参考第5章）。但是，我们的代码不是问是否，而是确定是否应该再执行一次代码。我们的代码会问一个问题，决定重复多少次代码块。这就是所谓的迭代。

6.2 “while” 循环，你真正需要的只有循环

这里有三个类型的循环，while循环，do-while循环，for循环。对于一开始我们需要聚焦到while循环（对不起，这个太有意思了）。对于一件事情，你真正的需要的循环是while。for循环，正如我们将看到的，仅仅是一个简单的代替计数操作的代替品，一个伟大的速记。Do-while很少使用（这本书中的例子没有一个需要它），所以我们将忽略它。

就像条件语句（if/else）的结构，while循环采用了一个布尔测试条件。如果测试的结果是真，执行大括号中的指示；如果测试的结果是假，我们将继续到下一行代码中。这里的区别是，如果是真，代码会不断的循环执行，直到测试的结果变成假。如图6.2所示。

图 6.2



让我们从腿的问题下手。假设下面的变量...

```

int y = 80;           // Vertical location of each line
int x = 50;           // Initial horizontal location for first line
int spacing = 10;      // How far apart is each line
int len = 20;          // Length of each line
  
```

我们不得不手动重复下面的代码：

```

stroke(255);
line(x,y,x,y + len); // Draw the first leg

x = x + spacing; // Add "spacing" to x
line(x,y,x,y + len); // The next leg is 10 pixels to the right

x = x + spacing; // Add "spacing" to x
line(x,y,x,y + len); // The next leg is 10 pixels to the right

x = x + spacing; // Add "spacing" to x
line(x,y,x,y + len); // The next leg is 10 pixels to the right

// etc. etc. repeating with new legs
  
```

现在，随着我们学到了while循环这个知识，我们能够改写例6-3，添加一个变量告诉我们循环到什么时候停止，就像这样，在哪个像素停止。

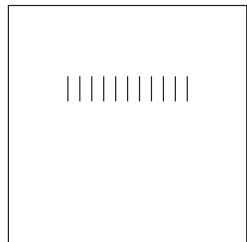
例 6-3：while循环

```
int endLegs = 150;
stroke(0);
while (x < endLegs) {
    line (x,y,x,y + len);
    x = x + spacing;
}
```

一个变量纪录了腿终止的坐标。

在while中循环绘制腿

图 6.3



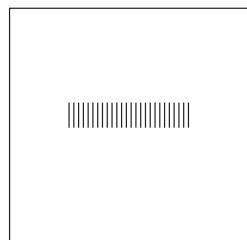
这样，我们代替了复杂的“`line(x,y,x,y + len);`”很多次我们都是第一次做，我们现在只需要将它放进while循环中就可以了，并且说：“只要x是小于150，在x上绘制一条线，并且同进行递增。”看看之前需要21行代码，现在只需要4行代码！

此外我们可以更改间距变量产生更多的腿。结果如图6.4所示。

```
int spacing = 4;
while (x < = endLegs) {
    line (x,y,x,y + len); // Draw EACH leg
    x = x + spacing;
}
```

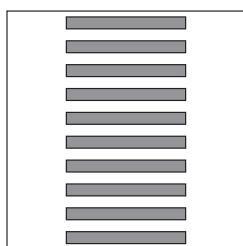
一个更小的间隔变量让腿与腿之间更近。

图 6.4



让我们看看其他例子，这次我们使用的是矩形而不是线，如图6.5所示，并且提出了3个关键问题。

图 6.5



1.你的循环初始条件是什么？在这里，第一个矩形的y坐标在10,我们想从10开始。

```
int y = 10;
```

2.什么时候你的循环需要停下来？我们想让矩形从窗口顶部循环，到底部停止，当它的y值大于那个高度我们就停止。换句话说，我们只需要小于我们设定的y的底部坐标值，它就会在那之前循环。

```
while (y < 100) {
    // Loop!
}
```

3. 你要循环的是什么东西？在这里，每次循环我们都想要一个新的矩形出现在先前的矩形下面。我们可以通过使用rect()函数以及20的递增做到这一点。

```
rect(100,y,100,10);
y = y + 20;
```

把它们放在一起。

```
int y = 10;
```

初始条件。

```
while (y < height) {
    rect(100,y,100,10);
```

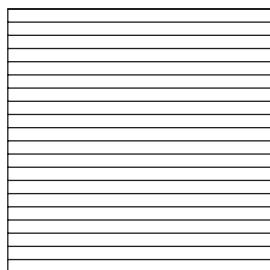
当布尔值是真，循环会继续。当布尔值是假，循环会停止。

```
}  
y = y + 20;
```

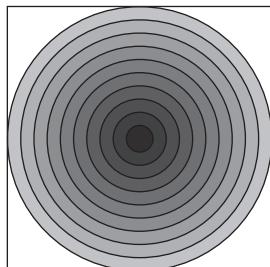
我们会以一个递增的y进行每次循环，一个接一个的绘制出矩形直到大于设定的高度就停止。



练习6-1：按照下列右边的截图，填写下列左边代码中的空白。



```
size(200,200);
background(255);
int y = 0;
while (_____){
    stroke(0);
    line(_____,_____,_____,_____);
    y = _____;
}
```

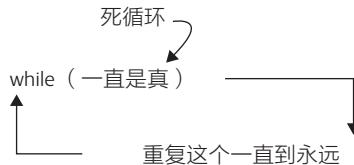


```
size(200,200);
background(255);
float w = _____;
while (_____){
    stroke(0);
    fill(_____);
    ellipse(_____,_____,_____,_____);
    _____ 20;
}
```

6.3 “退出”条件

循环可能让你开始意识到特别方便。不过在循环的世界也有黑暗的一面，它们会令人讨厌，并且被称为无限循环。如图6.6所示。

图 6.6



检查例6-3“腿”，我们可以看到只要x大于150时，它的循环就会停止。但是会一直发生，因为x有一个间距。它一直是一个整数。这不是一个事件，当我们确保循环发生时，也要确保循环能够很好的停止下来。

当你循环的退出条件没有时，Processing也不会检查出来并提醒你差一个退出条件。结果就是循环会一直不断的重复，不会停止。

例 6-4：无限循环，不要这么做！

```
int x = 0;

while (x < 10){
    println(x);
    x = x - 1;
}
```

x会进行无限的递减，因为在里x的值永远不会大于或等于10。一定要注意！

你可以尝试运行以上的代码（在运行之前你要确定你已经存储了你的重要文件，并且没有关键软件在运行）。你会很快看到Porcessing会卡住。唯一的解决办法就只能强制退出Processing。无限循环一般在编写代码中不会出现的那么明显，这里是另外一个有问题的编程，它有时会导致无限循环并崩溃。

例 6-5：另一种无限循环，不要这么做！

```
int y = 80;           // Vertical location of each line
int x = 0;            // Horizontal location of first line
int spacing = 10;     // How far apart is each line
int len = 20;          // Length of each line
int endLegs = 150;    // Where should the lines stop?

void setup() {
    size(200,200);
}

void draw() {
    background(0);
    stroke(255);
    x = 0;
    spacing = mouseX / 2;
```

在每条线之间的距离被设置成一个空间变量，
它被分配等于mouseX除以2。

```

while (x <= endLegs) {
    line(x,y,x,y + len);
    x = x + spacing;
}

```

退出条件 — 当x大于endlegs。

x是增量，x总是随着spacing而递增。但是间距可能的取值范围是什么？

会有一个无限循环发生？如果x从来没有大于150，我们将陷入一个无休止的循环。从x开始以spacing的大小递增，如果spacing是零。（或者是一个负数）x就会一直保持相同的值（或者负值）。

回顾我们在第4章中所学的constrain()函数，我们能够约束spacing能在一个正数值范围内，这样就不会出现无限循环。

```
int spacing = constrain(mouseX/2, 1, 100);
```

使用constrain()确保能够满足我们的退出条件。

由于spacing是直接联系到退出条件的关键因素，我们执行的时候要确保它的值不会导致无限循环的出现。换句话说，用我们的正常话说：“绘制了一系列的线，它们之间的距离为n，n不能小于1。”

这同样也是一个很实用的例子，因为它反应了关于mouseX有趣的事。你可能想吧mouseX的递增直接表达如下：

```

while (x <= endLegs) {
    line(x,y,x,y + len);
    x = x + mouseX / 2;
}

```

即使在循环内部放置mouseX，也不能解决无限循环的问题。

即使循环卡住了，只要用户只要把鼠标放在大于0的水平位置还是不能解决无限循环的问题。虽然这是一个很好的想法，但是它是有缺陷的。mouseX和mouseY的值在每次draw()循环的周期中都会更新新的值。因此，即使用户将鼠标位置移动到50，mouseX也永远不会直到这个新的值，因为它还是会卡在无限循环的那个值，并进行无限循环，无法到达draw()的下一个周期。

6.4 “for” 循环

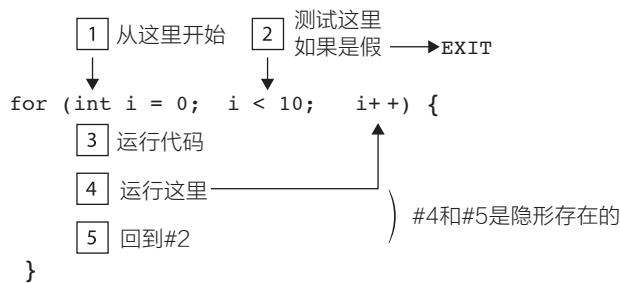
while循环是重复递增值的一个特别常见的风格（6.2节中演示）。一旦我们看看第9章数组，它会显得更加常见。for循环是一个这种类型的极好的快捷方式。在进入深化之前，让我们来谈谈一些在Processing中可能会出现的循环以及如何把它们写成一个for循环。

从0递增到9	for (int i = 0; i < 10; i = i + 1)
以10为间隔从0递增到100	for (int i = 0; i < 101; i = i + 10)
以5为间隔从100递减到0	for (int i = 100; i >= 0; i = i - 5)

看看以上的例子，我们能发现for循环有3个相同的部分：

- **初始化** — 在这里，声明并且初始化一个变量用于循环。这个变量大部分使用在循环内部作为一个开头。
- **布尔测试** — 这和我们在条件语句以及while循环语句中的布尔测试是一样的。它表达任何结果的方式只有真或假。
- **迭表达式** — 最后一个部分是在每次循环中发生的指令。请主义，该执行指令会在每次循环周期之后执行。（你可以有多个迭表达式，以及变量初始值，但是我们现在不用关心这些问题）

图 6.7



在英语中上述的代码意思是：重复这些代码10次。或者把它更简单的翻译成：计数从0到9。

在计算机语言中，意思是：

- 声明一个变量i，并且设置其初始值为0。
- 当i小于10的时候，循环这些代码。
- 在每次迭代结束时，添加1到i。

为了计数一个for循环可以有其自己的变量。没有在代码顶部中声明的变量称为局部变量。我们会不久后详细的解释它。

递增/递减运算符

在变量上加上或减去1的快捷方式，运算符号：

$x++$; 等同于: $x=x+1$; 意思是“递增的值为1”或者说“增加1到现在的变量x中”

$x--$; 等同于: $x=x-1$;

我们同样也有：

$x+=2$; 等同于: $x=x+2$;

$x*=3$; 等同于: $x=x*3$;

等一系列的类似符号（加减乘除）

用while格式完成的相同的循环：

```
int i = 0;
while (i < 10) {
    i++;
    //lines of code to execute here
}
```

用for语句重写腿的代码就像这样：

例 6-6： for循环格式的腿的绘制

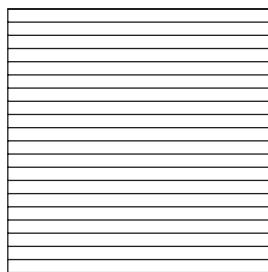
```
int y = 80;           // Vertical location of each line
int spacing = 10;     // How far apart is each line
int len = 20;          // Length of each line
```

```
for (int x = 50; x <= 150; x += spacing) {
    line(x,y,x,y + len);
}
```

把while循环的腿转译成for
循环的腿。

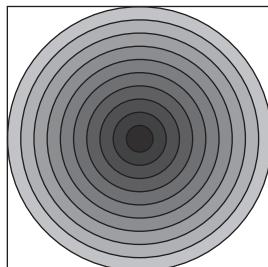


练习6-2：用for循环重新编写练习 6-2的代码



```
size(200,200);
background(255);

for (int y = _____; _____; _____) {
    stroke(0);
    line(_____,_____,_____,_____);
}
```

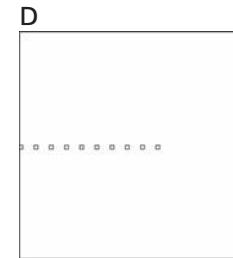
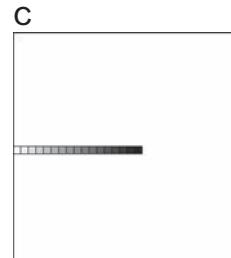
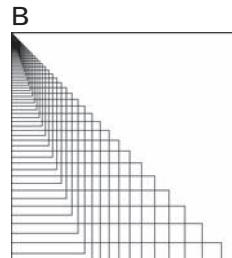
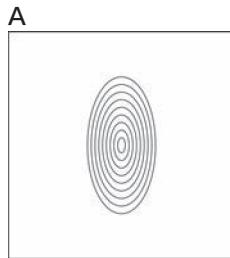


```
size(200,200);
background(255);

for (_____ : _____ : _____ - 20) {
    stroke(0);
    fill(_____);
    ellipse(_____,_____,_____,_____);
    ellipse(_____,_____,_____,_____);
}
```



练习6-3：以下是一些其他的循环例子。结合图中所示匹配合适的循环语句。它们有4行相同的初始代码。



```
size(300,300);           // Just setting up the size
background(255);         // Black background
stroke(0);               // Shapes have white lines
noFill();                // Shapes are not filled in
```

```
for (int i=0; i<10; i++) {
    rect(i*20,height/2, 5, 5);
}
```

```
int i=0;
while (i<10) {
    ellipse(width/2,height/2, i*10, i*20);
    i++;
}
```

```
for (float i=1.0; i<width; i+=1.1) {
    rect(0,i,i*2);
}
```

```
int x=0;
for (int c=255; c>0; c-=15) {
    fill(c);
    rect(x,height/2,10,10);
    x=x+10;
}
```

6.5 局部变量于全局变量（又名“变量作用域”）

直到现在，我们在编程的任何时候都会有变量，我们在声明的时候将它放在我们编程的顶部，即setup()的上面。

```
int x = 0;
```

我们经常会在我们编程的顶部声明变量。

```
void setup() {
}
```

这是一个很好的简化，并且它让我们方便于基本原则的声明，初始化和使用。但是变量也可以在编程的任何地方进行声明。现在我们将看看一个在顶部以外的位置声明变量意味这什么，并且看看怎么去正确的使用声明变量。

试想一下，电脑编程如果是你的生活。在生活中变量就好像是你要记住的一些数据。它有可能是一个你吃饭餐厅的地址。你把它在早晨记下来，然后去吃完饭之后会将它忘记。但是在另一个变量中它可能包含重要的信息（例如银行账户密码），你会单独将它保存在一个安全的地方。这是一个范围的概念。一些变量存在在整个编程中—全局变量；一些变量只是暂时的存在，只是为了在那一刻记住一些有价值的东西—局部变量。

局部变量是在一个代码块中声明的变量。到目前为止，我们已经看到了许多不同的代码块示例：setup()，draw()，mousePressed()，and keyPressed()，if语句以及while循环，for循环。

在代码块中声明的局部变量只适用于声明的那个代码中。如果你要在那个代码块外面使用这个局部变量，你会得到如下错误信息：

“No accessible field named “variableName” was found” (没有找到可访问的变量名称)

如果你没有声明变量的名称也会出现同样的错误信息。Processing不会知道存在的变量名称是什么如果你没有给它起名字。

下面是一个例子，它显示了在draw()中为了执行while循环而声明了局部变量。

例 6-7：局部变量

```
void setup() {
    size(200,200);
}
```

```
void draw() {
    background(0);
```

```
int x = 0;
while (x < width) {
    stroke(255);
    line(x,0,x,height);
    x += 5;
}
```

```
void mousePressed() {
    println("The mouse was pressed!");
}
```

x是不可用的！它是draw()中的局部变量。

x是可用的！因为它在draw()代码块中声明，所以在这里能够找到它。注意，在draw()上面的那块x是不可用的因为那里不属于draw()。同样，在while代码块中可以使用x，因为它也包含在draw()中。

x是不可用的！它是draw()中的局部变量。

你会想这是何苦呢？我们难道不能只把*x*作为一个全局变量吗？虽然你想的没错，但是它只是作用于draw()中，把它作为一个全局变量是一种浪费。有时候在编程时只在部分地方声明变量会更有效率，并且能够减少混乱。当然，有很多变量时全局的，但不是这里的情况。

for循环提供了一个局部变量以及初始化的内部环境：

```
for (int i = 0; i < 100; i += 10) {
    stroke(255);
    fill(i);
    rect(i,0,10,height);
}
```

虽然我们没有要求一定要在for循环中使用局部变量，但它通常是这样做比较方便。

理论上全局变量和局部变量名称可以相同。在这种情况下在局部变量范围内的会引用局部变量，在这个范围之外的就会用全局变量。在一般情况下，我们不建议用多个名称相同的变量，这样是为了避免在编程的时候混乱。



练习6-4：预测一下下面两个方案的结果。并运行看看你预测的对不对。



```
//SKETCH #1: Global
"count"
int count = 0;

void setup(){
    size(200,200);
}

void draw(){
    count = count+1;
    background(count);
}
```



```
//SKETCH #1: Local
"count"

void setup(){
    size(200,200);
}

void draw(){
    int count=0;
    count = count + 1;
    background(count);
}
```

6.6 重复内部的主要循环

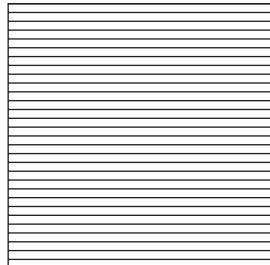
局部变量和全局变量之间的区别让我们离将Zoog成功地整合到一个循环结构中又近了一步。我们学完这1章之前，我想看看在你写的动态草图中一些常会出现地困惑点。

思考一下下面的循环（恰好是练习6-2的答案）。循环的结果如图6.8所示。

```
for (int y = 0; y < height; y += 10) {
    stroke(0);
    line(0,y,width,y);
}
```

比方说我们想采用上述的循环，并显示每一行一根线，让我们看到线条出现从上到下的动画。并我们首先想道德可能是把这个循环纳入带有setup()和draw()的动态草图中。

图 6.8



```
void setup() {
    size(200,200);
}

void draw() {
    background(255);
    for (int y = 0; y < height; y+=10) {
        stroke(0);
        line(0,y,width,y);
    }
}
```

如果我们阅读代码，它看起来似乎很有道理，我们会看到每行都会在一个时间点出现一条线。“设置一个尺寸为200*200的窗口。画一个白色背景。绘制一条y等于0的线。绘制一条y等于10的线。绘制一条y等于20的线。”

但是回顾第2章，我们还记得Processing直到draw()运行完之后才会重新运行。这对于我们在使用while和for循环的时候这个非常重要。这个循环会在同一个draw()周期中循环。draw()才是草图中的主循环。

如果要每次显示一条线，我们可以使用全局变量结合带有循环性质的draw()来使用。

例 6-8：每次显示增加一条线

int y = 0;

这里不要for循环，用一个全局变量代替。

```
void setup() {
    size(200,200);
    background(0);
    frameRate(5);
}
```

放缓帧速率让我们能够更容易看到显示出来的效果。

```

void draw() {
    // Draw a line
    stroke(255);
    line(0,y,width,y);
    // Increment y
    y += 10;
}

```

通过draw(), 每次增加一条线。

这个草图的逻辑和例4-3是相同的，我们的第一个变量动态草图。在这里我们移动的是一条线。（但不会在每次draw()循环后清除背景）

 练习 6-5：其实能够使用 for 循环实现一次出现一条线。如果你能弄清楚其中的原理就借用下面的代码试一试。

```

int endY;

void setup() {
    size(200,200);
    frameRate(5);
    endY = _____;
}

void draw() {
    background(0);
    for (int y = _____; _____; _____) {
        stroke(255);
        line(0,y,width,y);
    }
    _____;
}

```

在 draw() 中使用循环语句开辟了互动性的可能。例 6-9 显示了一系列举行（从左至右），其与鼠标之间的距离反映了每一个的亮度。

例 6-9：每次显示一条线

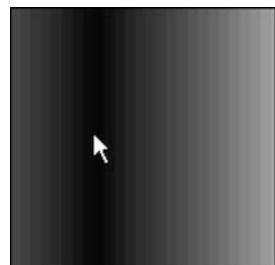
```

void setup() {
    size(255,255);
    background(0);
}

void draw() {
    background(0);
    // Start with i as 0
    int i = 0;
    // While i is less than the width of the window
    while (i < width) {
        noStroke();

```

图 6.9



```

float distance = abs(mouseX - i);

fill(distance);
rect(i,0,10,height);

// Increase i by 10
i += 10;
}
}

```

当前的矩形与鼠标之间的距离(distance)等于mouseX与i之间的差异的绝对值。

距离(distance)用于填充矩形的颜色。



练习6-6：用for循环重新编写一次例 6-9

6.7 Zoog 生长的手臂

我们上次用Zoog是让他在Processing窗口中来回弹跳。这个新版本的Zoog配备了一个小的变化。例6-10使用了一个for循环给Zoog添加了一些线，看起来像手臂。

例 6-10：Zoog的手臂

```

int x = 100;
int y = 100;
int w = 60;
int h = 60;
int eyeSize = 16;
int speed = 1;

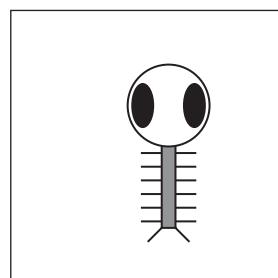
void setup() {
    size(200,200);
    smooth();
}

void draw() {
    // Change the x location of Zoog by speed
    x = x + speed;
    // If we've reached an edge, reverse speed (i.e. multiply it by -1)
    //((Note if speed is a + number, square moves to the right,- to the left)

    if ((x - width)(x < 0)) {
        speed = speed * -1;
    }
    background(255); // Draw a white background
    // Set ellipses and rects to CENTER mode
    ellipseMode(CENTER);
    rectMode(CENTER);
    // Draw Zoog's arms with a for loop
    for (int i = y + 5; i < y + h; i += 10) {
        stroke(0);
        line(x-w/3,i,x + w/3,i);
    }
}

```

图 6.10



用for循环画一系列的手臂。

```

// Draw Zoog's body
stroke(0);
fill(175);
rect(x,y,w/6,h*2);

// Draw Zoog's head
fill(255);
ellipse(x,y-h/2,w,h);

// Draw Zoog's eyes
fill(0);
ellipse(x-w/3,y-h/2,eyeSize,eyeSize*2);
ellipse(x + w/3,y-h/2,eyeSize,eyeSize*2);

// Draw Zoog's legs
stroke(0);
line(x-w/12,y + h,x-w/4,y + h + 10);
line(x + w/12,y + h,x + w/4,y + h + 10);
}

```

我们也可以利用一个循环实现多个Zoog，只需要将Zoog的身体放进for循环中就行。如例6-11所示。

例 6-11：多个Zoog

```

int w = 60;
int h = 60;
int eyeSize = 16;

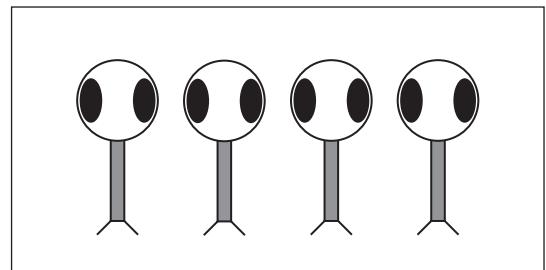
void setup() {
    size(400,200);
    smooth();
}

void draw() {
    background(255);
    ellipseMode(CENTER);
    rectMode(CENTER);
    int y = height/2;
    // Multiple versions of Zoog

    for (int x = 80; x < width; x += 80) {
        // Draw Zoog's body
        stroke(0);
        fill(175);
        rect(x,y,w/6,h*2);
        // Draw Zoog's head
        fill(255);
        ellipse(x,y-h/2,w,h);
        // Draw Zoog's eyes
        fill(0);
        ellipse(x-w/3,y-h/2,eyeSize,eyeSize*2);
        ellipse(x + w/3,y-h/2,eyeSize,eyeSize*2);
        // Draw Zoog's legs
        stroke(0);
        line(x-w/12,y + h,x-w/4,y + h + 10);
        line(x + w/12,y + h,x + w/4,y + h + 10);
    }
}

```

图 6.11



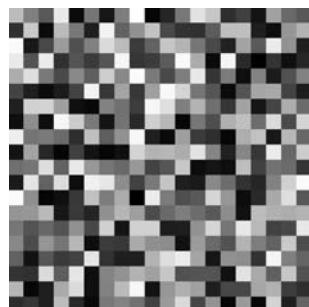
x在for循环中，为了能够重复循环Zoog。



练习 6-7：添加一些你自己设计的东西到循环中试一试。看看有什么新奇的东西？



练习6-7：使用 `for` 循环创建一个方形网格（每个颜色随机）。（提示：你需要 2 个`for`循环）然后重新用相同的模式用 `while` 循环进行编程。





第二课项目

- 步骤1** 将你在第一课设计的东西用变量进行重写。思考在你的设计中创建一个for循环。
- 步骤2** 编写一系列赋值操作，改变这些变量的值并让它们动起来。你还能使用系统变量，如 width , height, mouseX 和 mouseY。
- 步骤3** 在基于你的设计上使用条件语句。如果碰到边缘是什么情况，或者它放大到一定大小又是什么情况。如果鼠标移动到你设计的元素上会发生什么样的事。

第三课

组织

7. 函数

8. 对象

7. 函数 (functions)

“把一切合起来比分解它们更好。”

— Tears for Fears

本章内容：

- 模块化
- 声明和定义一个函数
- 调用函数
- 参数传递
- 返回一个值
- 重复使用性

7.1 分解它们

在第1章到第6章中提供的例子都是很短的。我们可能还没看过超过100行的代码。这些编程就像是这一章的段首，而不是整章内容。

Processing是很强大的，因为我们能用很少的代码作出很有趣的视觉表现。但是如果我们要向前迈进做更复杂的项目时，如网络应用程序或者图像处理时，我们可能会有几百行代码。我们会写成散文一样，没有段落。这些代码会在2个主要的代码块setup() 和draw() 这是非常笨重的。

函数是一种将我们的编程打包以及模块化的手段，这样能让我们的代码更易读，更易修改。让我们想一想视频游戏太空侵略者。我们在draw()中的步骤就好像：

- 擦除背景
- 绘制宇宙飞船
- 绘制敌人
- 根据用户使用键盘移动宇宙飞船
- 移动敌人

关于它的名字

函数经常会有其他叫法，如“Procedures”(方式)或“Methods”(方法)或者“Subroutines”(子程序)。在某些编程语言中它们是有严格区分的，“Procedures”是指执行一个任务，函数是指计算值。在这一章中，为了简单起见我使用函数这个名称。然而在Java编程语言中这个技术术语叫作“method”(面向对象的设计)，一旦我们进入到第8章，我们会使用“method”来描述在对象内部的函数。

在学习这章的函数之前，我们需要把上面的伪代码转译成实际的代码，并且放到draw() 里面。接着，我们就会面临如下问题：

```
void draw() {  
    background(0);  
    drawSpaceShip();  
    drawEnemies();  
    moveShip();  
    moveEnemies();  
}
```

我们会在 draw() 中调用这些函数。

上面的部分告诉我们如何将代码分开不同组方便编程以及管理。尽管如此，我们还缺少一个重要部分：函数的定义。调用函数是一个老方法。让我们想想每次编写 `line()`, `rect()`, `fill()` 等等的代码，并以此类推定义一个新的我们自己“虚构”的函数。

在我们进入详细学习之前，让我们想想为什么编写我们自己的函数这么重要：

- **模块化** — 函数将一个庞大的编程奉分解成一些小的部分，这样更利于阅读和管理。一旦我们已经想通了绘制飞船的方法，我们就可以将绘制飞船的这组代码块存在一个函数中，并且在需要的时候拿出来（调用的时候不用写出全部细节代码）。
 - **重复使用性** — 函数允许重复的使用并且无需重新编写整块代码。如果我们想让2个玩家每人都有一个太空飞船怎么办？我们可以重复使用drawSpaceShip()函数，对它进行多次调用，无需重写代码。

在本章中，我们将看到一些我们以前的编程，没有我们自己逼的函数，以及模块化和可重复性的函数。此外我们还会进一步强调局部变量和全局变量之间的区别，函数是一个单独的代码块，它需要使用到局部变量。最后我们将继续使用Zooq。



练习 7-1：在下面表中写出你的答案

7.2 “用户自定义”函数

在Processing中我们一致在使用函数。当我们写“`line(0,0,200,200);`”的时候，我们就调用了`line()`函数，它是Processing环境中的一个内置函数。通过调用函数`line()`来画一条线不是凭空存在的，这一般都是有人在什么地方（即在底层代码）自定义了它的结构。Processing的强项之一就是它的可用函数库，这是我们在这本书的前6章所探索的内容。现在是时候跨出内置函数的门，开始写属于我们自己的自定义函数（又名“虚构函数”）。

7.3 定义一个函数

函数的定义（有时候也被称为“声明”）有三个部分：

- 返回类型
- 函数名称
- 参数

它看起来就像这样：

```
returnType functionName (arguments) {
    // Code body of function
}
```

似曾相识？

还记得在第3章中我们介绍函数`setup()`和`draw()`吗？注意他们都遵循一套相同的格式，而这套格式就是我们现在在学的。

`setup()`和`draw()`是已经被定义了的函数，并且为了能够运行草图我们会从Processing中调用他们。其他我们自己写的函数必须由我们自己调用。

现在让我们仅仅关注与函数名和代码本身，忽略返回值和参数。

这里是一个简单的例子：

例 7-1：定义一个函数

```
void drawBlackCircle() {
    fill(0);
    ellipse(50,50,20,20);
}
```

这是一个简单的函数用于执行一个基本的任务：画一个黑色的圆，坐标是(50,50)。它的名字是一— drawBlackCircle() — 名字可以任意取，它的代码包括2个结构（我们尽可能选择了简短的代码）。同样这也是提醒我们它仅仅是一个函数的定义。除非这个函数会在编程的某处被调用并且执行，否则这个函数永远不会被显示。也就是说它需要饮用函数的名称，调用函数，如例7-2所示。

例 7-2：调用函数

```
void draw() {
    background(255);
    drawBlackCircle();
}
```

练习7-2：写一个 Zoog 的函数（或者你自己设计的什么东西）。并且从 draw() 中调用它。



```
void setup() {
    size(200,200);
}
void draw() {
    background(0);

    _____
    _____
    _____
    _____
    _____
    _____
```

7.4 简单的模块化

让我们来看看第5章小球弹跳的例子，并且用函数重编写它。用新的模块化分解他们。为了方便，例5-6这里重写了一边。

例 5–6：弹弹球

```
// Declare global variables
int x = 0;
int speed = 1;

void setup() {
  size(200,200);
  smooth();
}

void draw() {
  background(255);

  // Change x by speed
  x = x + speed;

  // If we've reached an edge, reverse speed
  if ((x > width) || (x < 0)) {
    speed = speed * -1;
  }

  // Display circle at x location
  stroke(0);
  fill(175);
  ellipse(x,100,32,32);
}
```

让球动起来

让球弹起来

让球显示出来

一旦我们确定我们怎么分割这些代码到函数中去，我们就可以将代码放到分类的函数中，将定义的函数放到 draw() 中，在 draw() 调用函数。函数通常放到 draw() 下面。

例 7–3：函数弹弹球

```
// Declare all global variables (stays the same)
int x = 0;
int speed = 1;

// Setup does not change
void setup() {
  size(200,200);
  smooth();
}

void draw() {
  background(255);
  move();
  bounce();
  display();
}
```

代替了复杂的编码，我们用简单的3个函数来绘制弹弹球。我们怎么知道这些函数的名称的呢？他们的名称就是我们创造的！

```
// A function to move the ball
void move() {
    // Change the x location by speed
    x = x + speed;
}

// A function to bounce the ball
void bounce() {
    // If we've reached an edge, reverse speed
    if ((x > width) || (x < 0)) {
        speed = speed * -1;
    }
}

// A function to display the ball
void display() {
    stroke(0);
    fill(175);
    ellipse(x,100,32,32);
}
```

你需要将这些函数放在哪里？

你能在除了**setup()**和**draw()**里面的任何位置定义函数。

但是为了方便，一般将你定义的函数放在**draw()**后面。

看见没，**draw()**已经变得很简单了。代码减少了函数的调用；详细的变量变化以及形状的显示都留给定义函数来完成。这就是程序员理智的最大优点。如果你在去加勒比海上旅游2周，回来后你一样能够很好理解这些代码。要改变球是如何呈现的你只需要编辑**display()**函数，不需要再在长长的代码中寻找那一行或者也不用担心修改成其他的位置了。例如，你想尝试更换**display()**如下：

```
void display() {
    background(255);
    rectMode(CENTER);
    noFill();
    stroke(0);
    rect(x,y,32,32);
    fill(255);
    rect(x - 4,y - 4,4,4);
    rect(x + 4,y - 4,4,4);
    line(x - 4,y + 4,x + 4,y + 4);
}
```

如果你想改变形状，你可以在**display()**函数中编写，并且草图其他部分的东西不会有任何改动。

使用函数的另一个好处就是方便调试。假设就在现在，我们的弹球函数没有按照我们所想的在运动。为了找到这个问题，我现在可以选择开启或者关闭这个部分的编程。例如我肯呢干只单独运行**display()**这个函数，而把**move()**和**bounce()**注释掉。

```
void draw() {
    background(0);
    // move();
    // bounce();
    display();
}
```

函数能够被放在注释里面，以方便我们确定是不是它出现错误。

被定义的函数**move()**和**bounce()**仍然存在，只是现在我们没有调用那个函数。通过一个一个的添加函数，我们能够很简单的排除有问题的代码。



练习7-3：将你所写的任何一个编程编程如上的模块化函数。使用下面的位置来创建你需要的函数列表。

7.5 参数

在几页之前，我们说：“让我们先忽略返回类型和参数。”这么做是为了让我们对函数的入门变的简单一些。然而，函数的能力绝不仅仅只是将编程分块。一键解锁这些强大能力的就是参数这个概念（又名“parameters”）！

参数的值会直接通过函数。你应该能想象到参数在什么情况下会在函数中。对于移动，一个函数可能会说“移动n步”，其中“n”就是参数。

当我们在Processing中显示了一个圆的时候，我们需要圆的详细参数。我们肯定不能只说我要一个圆，我们必须还得说圆得位置和尺寸。他们就是ellipse()函数得参数，我们早在第1章就碰到过了。

让我们重写一次drawBlackCircle()，让它包含参数：

```
void drawBlackCircle(in diameter){  
    ellipse(50,50,diameter,diameter);  
}
```

“diameter”就是drawBlackCircle()函数中的一个参数。

参数就是一个在函数定义括号内的简单的变量声明。这个变量是一个局部变量（还记得第6章我们讨论的东西吗？）它用于函数内部（并且只能用于函数内部）。白色圆的尺寸将根据括号内的变量决定。

```
drawBlackCircle(16); // Draw the circle with a diameter of 16  
drawBlackCircle(32); // Draw the circle with a diameter of 32
```

找到弹球的例子，我们将其中move()函数进行重写将其包括参数：

```
void move(int speedFactor) {  
    x = x + (speed * speedFactor);  
}
```

参数“speedFactor”直接影响圆移动的速度。

为了让球移动比之前移动快2倍：

```
move(2);
```

或者更快：

```
move(5);
```

我们也将其他变量或数学表达式（如mouseX除以10）放到函数中。如例子：

```
move(mouseX/10);
```

参数让能够重复使用的函数更灵活。为了证明这一点，我们将看看一个形状集合的代码，并研究让函数允许我们绘制多个版本的形状而无需重新输入一遍又一遍的代码。

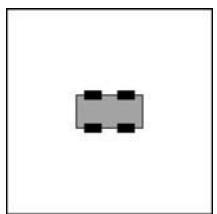
离开Zoog一会，思考一下下面类似汽车的模型（如图7.1所示）。

```
size(200,200);
background(255);
int x = 100; // x location
int y = 100; // y location
int thesize = 64; // size
int offset = thesize/4; // position of wheels relative to car

// draw main car body (i.e. a rect)
rectMode(CENTER);
stroke(0);
fill(175);
rect(x,y,thesize,thesize/2);

// draw four wheels relative to center
fill(0);
rect(x - offset,y - offset,offset,offset/2);
rect(x + offset,y - offset,offset,offset/2);
rect(x - offset,y + offset,offset,offset/2);
rect(x + offset,y + offset,offset,offset/2);
```

图 7.1



这个车需要5个矩形组成，一个大的矩形在中间，4个轮子矩形在四周。

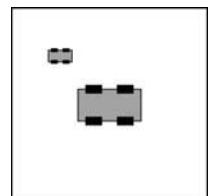
让我们继续绘制第二个车，用跟上面同样的代码，但是值不同，如图7.2所示。

```
x = 50; // x location
y = 50; // y location
thesize = 24; // size
offset = thesize/4; // position of wheels relative to car

// draw main car body (i.e. a rect)
rectMode(CENTER);
stroke(0);
fill(175);
rect(x,y,thesize,thesize/2);

// draw four wheels relative to center
fill(0);
rect(x - offset,y - offset,offset,offset/2);
rect(x + offset,y - offset,offset,offset/2);
rect(x - offset,y + offset,offset,offset/2);
rect(x + offset,y + offset,offset,offset/2);
```

图 7.2



在画第二个车时每一个代码都重
复了。

这个问题看起来是相当明显的。毕竟，我们都在做两次相同的事，为什么我们还要重复所有的代码呢？为了拜托这种重复，我们需要通过几个不同的参数将代码转换到中（位置，尺寸，颜色）。

```
void drawcar(int x, int y, int thesize, color c) {
    // Using a local variable "offset"
    int offset = thesize/4;
    // Draw main car body
    rectMode(CENTER);
    stroke(200);
    fill(c);
    rect(x,y,thesize,thesize/2);

    // Draw four wheels relative to center
    fill(200);
    rect(x - offset,y - offset,offset,offset/2);
    rect(x + offset,y - offset,offset,offset/2);
    rect(x - offset,y + offset,offset,offset/2);
    rect(x + offset,y + offset,offset,offset/2);
}
```

可以在函数中声明一个局部变量并使用。

这些代码定义了这个函数。函数drawCar()基于四个参数绘制了一个车：横向坐标，纵向坐标，尺寸以及颜色。

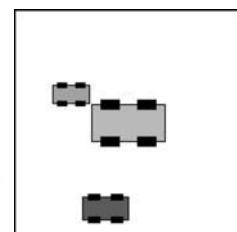
在draw()函数中，我们能够调用drawCar()函数三次，通过使用这4个参数。如图7.3所示。

图 7.3

```
void setup() {
    size(200,200);
}

void draw() {
    background(0);
    drawCar(100,100,64,color(200,200,0));
    drawCar(50,75,32,color(0,200,100));
    drawCar(80,175,40,color(200,0,0));
}
```

这个代码用正确的参数以及顺序，调用了3次函数。

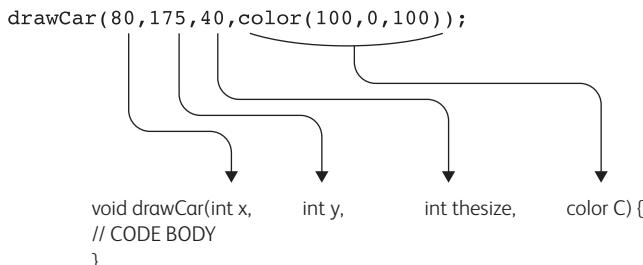


从技术上讲，参数都在函数的括号内被声明，就是“void drawCar(int x, int y, int thesize, color c).” 当函数被调用的时候参数的值会通过这些变量进行传递，如“drawCar(80,175,40,color (100,0,100));”。

这种概念主要在于参数的转化传递。除非我们很熟悉这种技术，要不然我们不会提前进行这种编程。

让我们用文字说明。想象一下一个阳光明媚的日子，你和你的朋友在公园里玩耍。你有球。你（主程序）通过球（参数）调用了函数（你的朋友）。你的朋友（参数）现在有球，并且可以使用它。如图7.4所示。

图 7.4



关于参数传递一些重要的内容：

- 你必须在函数中定义这个相同数量的参数。
- 当一个参数被传递时，它的类型必须和函数定义中声明的参数是一个类型。整数必须对应到整数，浮点数必须对应到浮点数，以此类推。
- 作为你传递的参数的值可以是一个值(20, 5, 4.3, 等等)，一个变量(x, y, 等等)，或者一个表达式的结果(8 / 3, 4 * x/2, random(0,10), 等等)。
- 参数作为一个函数的局部变量，只能在那个函数中进行使用。

练习7-4：下面函数需要3个数字，让他们相加，并且让相加的结果显示在信息窗口中。



```
void sum(int a, int b, int c) {  
    int total = a + b + c;  
    println(total);  
}
```

看到上面的定义函数，写出调用它的代码。

练习7-5：OK,这是一个相反的问题。这里需要假定一个函数需要2个数字让他们相乘，并且结果显示在信息窗口中。写出这个定义函数。



```
multiply(5.2,9.0);
```



练习7-6：这里是例5-6的弹弹球结合了 drawCar() 函数。填写空报让你现在有一个参数传递的弹弹球！（注意为了区分开 drawCar() 中的局部变量x,y；全变量现在叫作 globalX 和 globalY ）。

```
int globalX = 0;
int globalY = 100;
int speed = 1;

void setup() {
    size(200,200);
    smooth();
}

void draw() {
background(0);

}

void move() {
    // Change the x location by speed
    globalX = globalX + speed;
}

void bounce() {
    if ((globalX > width) || (globalX < 0)) {
        speed = speed * -1;
    }
}

void drawCar(int x, int y, int thesize, color c) {
    int offset = thesize / 4;
    rectMode(CENTER);
    stroke(200);
    fill(c);
    rect(x,y,thesize,thesize/2);
    fill(200);
    rect(x - offset,y - offset,offset,offset/2);
    rect(x + offset,y - offset,offset,offset/2);
    rect(x - offset,y + offset,offset,offset/2);
    rect(x + offset,y + offset,offset,offset/2);
}
```

7.6 传递的复制

这里有一个小问题，我们拿“玩耍”作为类比。我真正说的应该如下。在折腾球（参数）之前，我们为它做了一个副本（第二个球），并把它传递给接收器（函数）。

每当你传递一个初始值到函数的时候，你实际上不是传递这个值本身，而是传递它的变量副本。早传递硬编码数字的时候这看起来区别很小，但是当传递变量的时候就显得很不一样了。

下面有一个代码名为 `randomizer()` 的函数，设置了一个参数（浮点数）以及，会加上-2到2之间的随机数。这是伪代码。

- num是10
- num显示10
- 一个数字通过参数newnum传递到函数 `randomizer()` 中。
- 在函数`randomizer()`中：
 - 一个随机的数字加上newnum
 - newnum显示10.34232
- num显示：一直是10！一个副本被传送到newnum，但是num并没有改变。

这是实际代码：

```
void setup() {
  float num = 10;
  println("The number is: " + num);
  randomizer(num);
  println("The number is: " + num);
}

void randomizer(float newnum) {
  newnum = newnum + random(-2,2);
  println("The new number is: " + newnum);
}
```

即使变量num转换到变量newnum中去了，但是原来的值并没有收到任何改变，因为它就像创造了一个副本。

我喜欢称这个过程叫作“通副本”，但是通常简称为“传递值”。这包括了原有数据类型，但是不适用于我们下一章中关于对象的情况。

这个例子也给了我们一个很好的机会来查看当我们在使用函数时程序的流程。注意代码的执行线路，当函数被调用的时候，代码会离开当前的那一行执行那个被调用的函数，然后回到之前离开的地方。下面是上面例子流程的描述：

1. 设置num等于10
2. 显示num的值
3. 调用函数randomizer
 - a. 设置newnum等于newnum加上一个随机数
 - b. 显示newnum
4. 显示num的值

练习7-7：预测一下下面所写的代码会显示什么在窗口中。



```

void setup() {
  println("a");
  function1();
  println("b");
}

void draw() {
  println("c");
  function2();
  println("d");
  function1();
  noLoop();
}

void function1() {
  println("e");
  println("f");
}

void function2() {
  println("g");
  function1();
  println("h");
}

```

Output:

新东西！noLoop()是Processing中的一个内置函数，它能停止draw()中的循环。在这种情况下我们能确保draw()只运行一次。我们在代码中其他位置使用函数loop()能够重新启动循环。

从一个函数内调用另一个函数是完全合理的。实际上我们每次在setup()或者draw()中调用函数也是同样的情况。

7.6 返回类型

到目前为止，我们已经看到了函数是如何分成一部分一部分的来做，并且结合参数能够重复使用。但是仍然还缺少一块，我想你肯定想知道：void(无效)是什么意思？

```
ReturnType FunctionName ( Arguments ) {
    //code body of function
}
```

作为一个提示，让我们来看看定义一个函数的架构：

```
// A function to move the ball
void move(int speedFactor) {
    // Change the x location of organism by speed multiplied by speedFactor
    x = x + (speed * speedFactor);
}
```

ok，让我们来看看我们的一个函数：

“move”是函数名称，“speedFactor”是函数的一个参数，“void”是返回类型。到现在为止我们定义的所有函数都没有返回类型，这就是“void”的意思：没有返回类型。但是什么是返回类型以及我们什么时候需要它呢？

让我们回忆一下在第四章中研究的 random() 函数。我们需要一个0到某些数之间的一个随机值的这种函数，random() 能够很好的执行我们所需要的这些内容。random()函数返回了一个值。什么类型的值？一个浮点数。在 random() 中的返回类型是浮点数。

返回类型是函数返回的数据类型。在 random() 中，我们没有指定返回类型，然而 Processing 的开发者已经做了这些，并且能够在参考页面看到 random() 的文件。

每次调用random()函数的时候，它都会返回一个范围中的随机值。如果有一个参数使用这个函数就会返回一个在0到那个参数之间的一个随机浮点数。函数调用 random(5) 会返回一个0到5之间的随机值。如果是2个参数，它会返回一个这2个参数之间的一个浮点数。函数调用random(-5, 10.2) 会返回一个-5到10.2之间的数。

来自 <http://www.processing.org/reference/random.html>

如果我们想自己写一个有返回值的函数，我们必须指定函数的返回了！行。让我们创建一个简单的例子吧：

```
int sum(int a, int b, int c) {
    int total = a + b + c;
    return total;
}
```

这个函数同时设置了3个参数，并且返回一个类型 - int.

这里需要一个返回语句！一个带有返回类型的函数必须始终返回该类型的值。

正如我们先前的例子，我们需要写一个返回类型 int。这定义了函数必须返回一个整数类型的数。为了能够返回这个值，必须有一个返回语句。返回语句就像这样：

```
return valueToReturn;
```

如果我们没有加入返回语句，那么Processing就会给予我们一个报错：

The method “int sum(int a, int b, int c);” must contain a return statement with an expression compatible with type “int.”

“int sum(int a, int b, int c);” 必须结合一个返回语句以搭配我们的返回类型“int。”共同使用。

只要返回语句被执行，编程就会退出函数，并且发送一个返回值到该函数代码的位置。该值可以用于赋值操作（给另一个变量）或者是任何其他表达方式。参见图7.5中的插画。这里有一些例子：

```
int x = sum(5,6,8);
int y = sum(8,9,10) * 2;
int z = sum(x,y,40);
line(100,100,110,sum(x,y,z));
```

我讨厌又在公园里面追赶，但是你可以想一想。你（主程序）仍球给你的朋友（一个函数）。之后你的朋友抓到球，他会思考一会，然后把球里面装一个数字（一个返回的值）然后传递回给你。

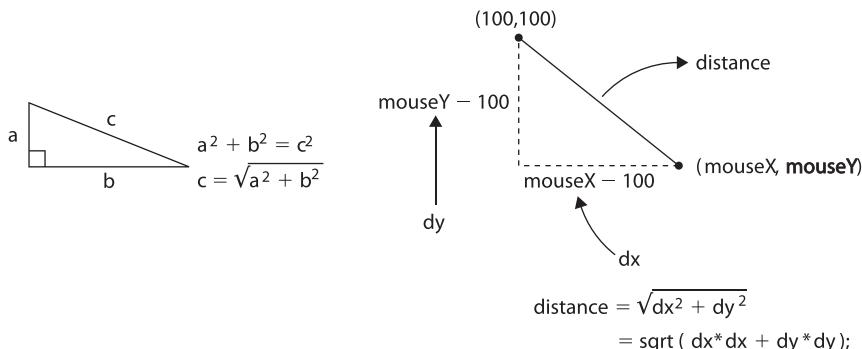
有返回值的函数一般用于执行复杂的计算，它可能在整个编程中需要多次执行。其中一个例子是计算2个点之间的距离：(x_1, y_1) 和 (x_2, y_2)。像素之间的距离在交互应用中是一个非常有用的部分。在Processing，实际上我们有内置函数可以直接使用，它被称为 dist()。

```
float d = dist(100, 100, mouseX, mouseY);
```

计算(100,100)与(mouseX, mouseY)
之间的距离。

上面代码计算了鼠标的位置到(100,100)之间的距离。现在让我们假装Processing中没有包括这个内置函数。没有它，我们不得不自己动手计算这个距离，通过勾股定理，如图7.6所示。

图 7.6



```
float dx = mouseX - 100;
float dy = mouseY - 100;
float d = sqrt(dx*dx + dy*dy);
```

如果我们想要这个计算在编程中可以多次执行，我们会设置一个函数，它的返回值为d。

```
float distance(float x1, float y1, float x2, float y2) {
    float dx = x1 - x2;
    float dy = y1 - y2;
    float d = sqrt(dx*dx + dy*dy);
    return d;
}
```

我们自己写的dist()函数版本。

注意，这里的返回类型是float。同样我们不需要再次写这个函数因为Processing本身就自带这个函数。但是为了更好的理解我们会单独的写这个函数作为学习。

例 7-4：使用函数返回一个值，距离

```
void setup() {
    size(200,200);
}

void draw() {
    background(0);
    stroke(0);

    // Top left square
    fill(distance(0,0,mouseX,mouseY));
    rect(0,0,width/2 - 1,height/2 - 1);

    // Top right square
    fill(distance(width,0,mouseX,mouseY));
    rect(width/2,0,width/2 - 1,height/2 - 1);

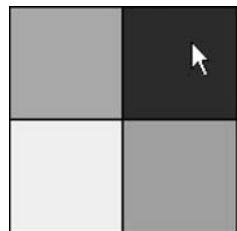
    // Bottom left square
    fill(distance(0,height,mouseX,mouseY));
    rect(0,height/2,width/2 - 1,height/2 - 1);

    // Bottom right square
    fill(distance(width,height,mouseX,mouseY));
    rect(width/2,height/2,width/2 - 1,height/2 - 1);
}

float distance(float x1, float y1, float x2, float y2){
    float dx = x1 - x2;
    float dy = y1 - y2;
    float d = sqrt(dx*dx + dy*dy);
    return d;
}
```

我们的距离函数用于计算每个像素的亮度值。我们可以使用内置函数dist()代替我们自己写的，但是我们正在学习如何写我们自己的函数。

图 7.7





练习7-8：写一个函数带有一个参数—F代表华氏度，将它转换称摄氏度并且计算出一个结果。

```
C (F - 32) * (5/9)
_____ tempConverter(float _____) {
    _____ = _____
    _____
}
```

7.6 Zoog 重组

Zoog 现在已经准备好大改造。

- 重组 Zoog 需要2个函数：drawZoog() 和jiggleZoog()。为了有意思，我们会有一个Zoog（在x和y方向随机移动）摇摆代替原来的弹跳。
- 纳入参数，让Zoog的摇动根据mouseX的位置决定，让Zoog眼睛的颜色根据Zoog距离鼠标的位置决定。

图 7.8

例 7-5：带有函数的Zoog

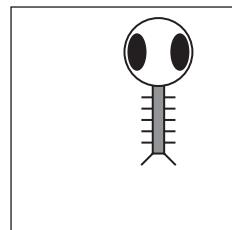
```
float x = 100;
float y = 100;
float w = 60;
float h = 60;
float eyeSize = 16;

void setup() {
size(200,200);
smooth();
}

void draw() {
background(255);      // Draw a black background

// mouseX position determines speed factor for moveZoog function
// float factor = constrain(mouseX/10,0,5);
jiggleZoog(factor);

// pass in a color to drawZoog
// function for eye's color
float d = dist(x,y,mouseX,mouseY);
color c = color(d);
drawZoog(c);
}
```



代码可以改变Zoog相关的变量并且显示Zoog在draw()之外移动以及调用内置函数。函数给定了一些参数，如“晃动Zoog有以下因素”和“用下面的眼睛颜色绘制Zoog”。

```
void jiggleZoog(float speed) {
    // Change the x and y location of Zoog randomly
    x = x + random(-1,1)*speed;
    y = y + random(-1,1)*speed;

    // Constrain Zoog to window
    x = constrain(x,0,width);
    y = constrain(y,0,height);
}

void drawZoog(color eyeColor) {
    // Set ellipses and rects to CENTER mode
    ellipseMode(CENTER);
    rectMode(CENTER);

    // Draw Zoog's arms with a for loop
    for (float i = y - h/3; i < y + h/2; i += 10) {
        stroke(0);
        line(x - w/4,i,x + w/4,i);
    }

    // Draw Zoog's body
    stroke(0);
    fill(175);
    rect(x,y,w/6,h);

    // Draw Zoog's head
    stroke(0);
    fill(255);
    ellipse(x,y - h,w,h);

    // Draw Zoog's eyes
    fill(eyeColor);
    ellipse(x - w/3,y - h,eyeSize,eyeSize*2);
    ellipse(x + w/3,y - h,eyeSize,eyeSize*2);

    // Draw Zoog's legs
    stroke(0);
    line(x - w/12,y + h/2,x - w/4,y + h/2 + 10);
    line(x + w/12,y + h/2,x + w/4,y + h/2 + 10);
}
```

 练习7-9：根据例6-11的版本（多个Zoog）调用一个函数绘制Zoog。定义出这个函数并且完成这个草图。你可以随意设计Zoog。

```
void setup() {  
    size(400,200); // Set the size of the window  
    smooth(); // Enables Anti-Aliasing (smooth edges on shapes)  
}  
  
void draw() {  
    background(0); // Draw a black background  
    int y = height/2;  
    // Multiple versions of Zoog are displayed by using a for loop  
    for (int x = 80; x < width; x += 80) {  
        drawZoog(x,100,60,60,16);  
    }  
}
```

 练习7-10：用函数重新写一遍第二课项目。

8. 对象 (Objects)

“没有对象是如此的美丽，在某些条件下，它也没那么难看。”

— Oscar Wilde

本章内容：

- 数据和功能，最后一起
- 什么是对象
- 什么是类
- 书写你自己的类
- 创建你自己的对象
- Processing的“标签(tabs)”

8.1 面向对象的编程

在我们开始研究如何进行面向对象编程(OPP)之前，让我们首先理解以下“对象”的概念。这里的重点是我们没有引入任何新的编程知识：对象所需要的一切我们都已经学习过：变量，条件语句，循环，函数等等。在这里什么是全新的？它是一种思维方式，将我们所学的东西进行构建以及组织。

想象以下你并不是在Processing编程，而是为了写出一个程序为了安排你的每天的时间，一个指令的列表，如果你试试，开始就会像这样：

- 起床
- 喝咖啡（或者茶）
- 吃早餐：麦片，蓝莓或者喝豆浆
- 乘坐地铁

这里涉及到了什么？具体来说有什么事情？首先它可能不会立即显示我们已经写在上面的指令，最主要的是你，一个人。你表现出某些特性。也许你有一头棕色头发，戴眼镜，并且像一个书呆子。你也有能力做一些事情，如起床（想必你也可以睡觉），吃东西，或者坐地铁。对象就是跟你一样的东西，有自身的属性并且能够做一些事情。

那么如何进行相关编程？对象的属性是变量，对象能做的事是函数。面向对象编程是结合了我们1到7章所学的一切，数据和功能全部加在一起。

让我们为一个简单的人体对象分配好数据和功能吧：

人体数据

- 高度
- 重量
- 性别

- 眼睛颜色
- 头发颜色

人体功能

- 睡觉
- 起床
- 吃饭
- 乘坐交通工具

现在，在我们进行更多学习之前，我们需要聊一个简单的话题。上述的结构不是人体本身，它只用来描述这个概念。它描述了什么是人。对人来说它有身高，头发，睡觉，吃饭等等。这时一个区别于编程对象关键的地方。这个人的模板被称为类。类不同与对象。你是一个对象，我是一个对象。爱因斯坦是一个对象。而人，我们所有的人是一个类（一个类别）。

想一想曲奇。一个曲奇能够制造饼干，但是不是饼干本身。曲奇是一个类，饼干是一个对象。

 练习 7-10：思考把车当作一个对象。一辆车有什么数据？有什么功能？

车的数据

车的功能

8.2 使用对象

在我们实际写一个类之前，让我们看一看我们如何在往程序里面使用对象（如 `setup()` 和 `draw()`）让编程变得更好。

返回第7章的汽车例子，你可能还记得它的伪代码可能像这样子：

数据(全局变量)：

汽车颜色

汽车的x坐标

汽车的y坐标

汽车的x方向速度

设置(setup):

初始化车的颜色

初始化车的位置以及起始点

初始化车速

绘制(draw):

填充背景

显示汽车的位置与颜色

显示汽车随车速度改变

在第7章中，我们在编程的顶部定义了全局变量，初始化内容在setup()，并且在draw()中调用函数以及显示汽车。

面向对象编程允许我们使用所有的变量以及主程序之外的函数并且能够将它们存储在一个汽车对象中。一个汽车对象必须有的一些数据—颜色，位置，速度。这时第一部分。第二部分是汽车对象能做的事，方法（一个对象里的函数），汽车能够移动并且显示出来。

使用面向对象编程设计之后，位代码可能会优化成下面这样：

数据(全局变量)

汽车对象

设置(setup):

初始化汽车对象

绘制(draw):

填充背景

显示汽车对象

移动汽车对象

注意我们在第一个例子中我们去掉了所有的全局变量。代替了独立的汽车颜色变量，汽车位置变量以及汽车速度变量，我们现在只有一个变量，一个汽车变量。并且只用初始化这一个汽车变量。这些变量去哪了？这些变量一直都还存在，知识现在它们在汽车对象中（我们在某一个时刻会在汽车类中定义它们）。

移开伪代码，我们的实际草图看起来就像这样：

```
Car myCar;

void setup() {
    myCar = new Car();
}

void draw() {
    background(0);
    myCar.move();
    myCar.display();
}
```

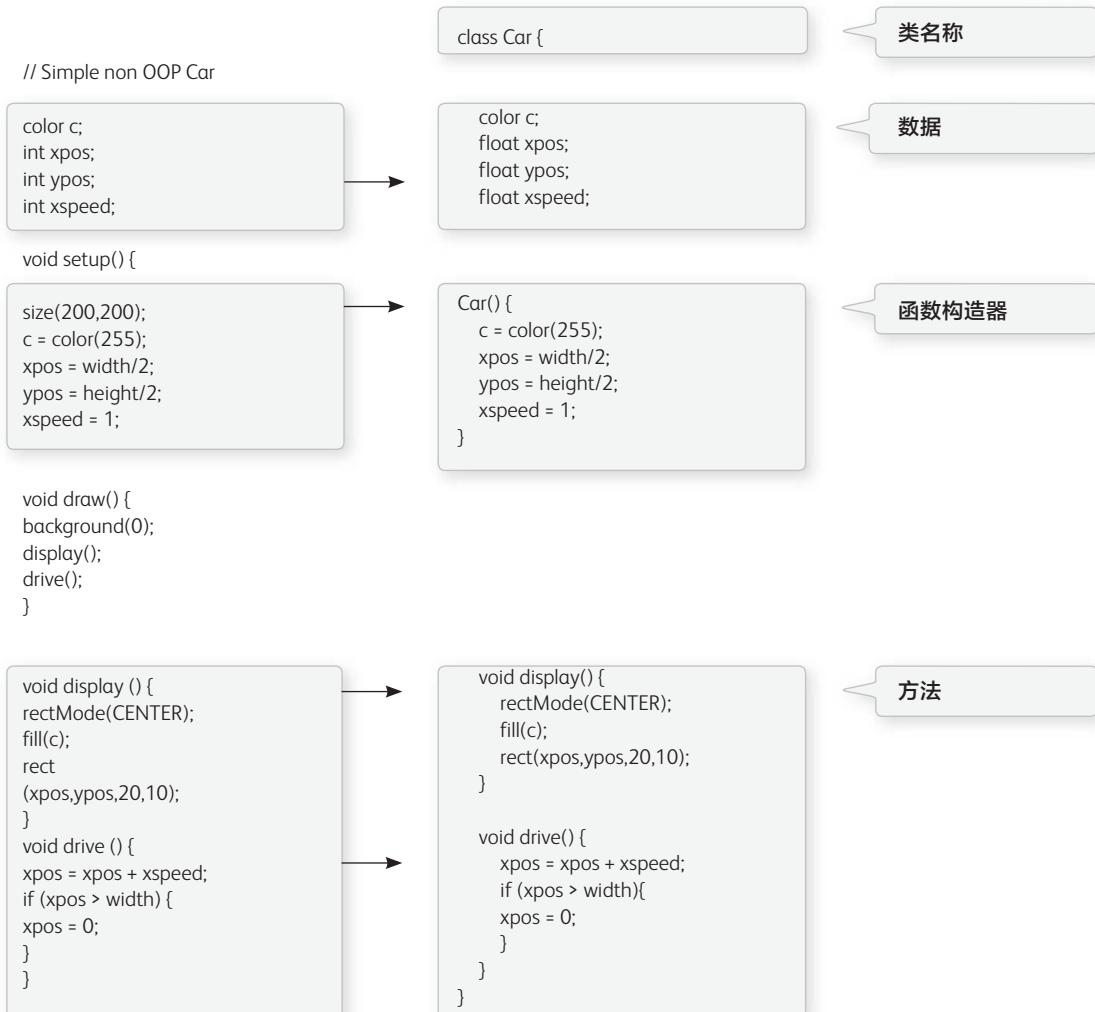
我们会提到很多之前学的相关代码，但是在这之前，让我们看看如何书写汽车类本身。

8.2 书写格式

上面的简单的汽车例子演示了在Processing中如何使用对象让代码更干净，更易读。但是难的工作就是书写对象的模板，也就是类。当你第一次学习面向对象编程，采用无对象，不改变函数的编写往往是一个有用的联系。我们将执行第七章汽车的例子，重新处案件一个外观和行为一样的面向对象。并且在本章结尾，我们将让Zoog重新编程一个对象。

所有的类必须包含4个元素：名称，数据，构造函数以及方法(从技术上说，只需要类的名称，但是对于整个面向对象编程是要包括这所有的元素)。

这里就是我们如何在一个简单的面向对象的草图中书写这些元素（一个简单的版本，如例7-6所示），并切把它们放入一个汽车类中，这样我们就能够创建汽车对象。



类名称

数据

函数构造器

方法

类名称—名称取决于你选择的类的这个名称。在声明名称之后，我们把所有代码放进它的大括号内。类的名称一般来说是首字母答谢（用于区别小写字母的变量名称）。

数据—数据就是这个类的变量的集合。这些变量通常被称为个体变量，因为每一个个体对象都包含变量。

构造—构造在一个类中是一个特殊的函数，它创建了对象本身。它给予设置对象提供指示。它就像Processing中的 `setup()` 函数，每当一个新的对象从类中创建出来它就用于创建一个单独的草图。它与类有同样的名称，并且调用的时候回称为一个新的运算符：`Car myCar = new Car();`

方法—我们可以通过书写方法添加一些函数在我们的对象中。这些都是和第7章用同样的方式进行描述，有返回类型，名称，参数，以及代码本身。

类的代码作为一个代码块能够放置在除了 `setup()` 以及 `draw()` 外的任何地方。

A Class Is a New Block of Code!

```
void setup() {
}

void draw() {
}

class Car {}
```

练习8-2：填写完整下列以人为类的声明。包括一个能够调用的函数 `sleep()` 或者创建你自己的函数。请根据汽车例子语法做。（这里没有以个完全正确或完全错误的答案，但是语句的结构很重要。）



```
color hairColor;
float height;
______() {
_____
}

} _____ {
_____
}

}
```

8.4 使用对象：详细

在章节8.2中，我们看到了对象如何让Processing草图更佳简化明确。

```
Car myCar;           第一步：声明一个对象

void setup() {        第二步：初始化对象
    myCar = new Car();
}

void draw() {          第三步：在对象里面调用函数
    background(0);
    myCar.move();
    myCar.display();
}
```

接下来让我们看看上面3个步骤的详细操作及使用。

步骤 1. 声明一个对象变量

如果你回到第4章，你可能还记得用一个类型和一个名称声明一个变量。

```
// Variable Declaration
int var; // type name
```

上面时声明变量一个最基本的例子，在这种情况下是整数。正如我们在第4章中了解的，原始数据类型包括：整数，浮点数，字符。声明一个变量和声明一个对象很相似。不同之处在于它的类型是类的名称，这是我们创建的，在这里叫“Car”，顺便说以下，这不是原始类型而是复杂的数据类型。（因为它储存了很多信息：数据以及函数。原始数据类型只储存数据）

步骤 2. 声明一个对象变量

同样，你可能还记得在第4章为了初始化变量（即给它一个初始值），我们绘使用一个参数操作—让变量等于某些东西。

```
// Variable Initialization
var = 10; // var equals 10
```

初始化对象相对复杂一些。它不是简单地将参数分配到一个原始类型值（如整数、浮点数等），我们需要构造一个对象。这时我们需要一个新的运算符new。

```
// Object Initialization
myCar = new Car();           使用new运算符创建一个新的对象
```

在上面例子中，“myCar”是对象名称，“=”表明我们设定它等于某些东西，那个东西就是一个新的汽车对象例子。我们在这里做的就是汽车对象的初始化。当你初始化一个原始类型变量，如整数时，你只需要设置它等于一个数字即可。但是一个对象可能结合了很多不同的数据。回顾我们上一章所讲的汽车类，我们看到有行代码在调用构造函数，一个叫Car()很特殊的函数，它初始化了所有的对象变量，并确保汽车对象已经准备好可以使用了。

这里还有另一件事，原始类型整数变量，如果你在初始化的时候忘记了（本来要设置它等于10），Processing绘给予它一个默认值，0。但是一个对象（如myCar）不会有默认值。如果你忘记初始化一个对象，Processing只会给予一个空值（即什么都没有）。不是0,也不是一个负数，就是一个空的东西。如果你在运行过程中下面窗口有错误信息显示“NullPointerException”（一个很常见的错误），这很大可能性就是你忘记初始化你的对象了（更多详细内容参见附录）。

步骤 3. 使用对象

一旦我们完成了对象的声明以及初始化，我们就可以使用对象了。使用一个对象涉及到该对象的内置函数。作为一个人的对象他能吃饭，作为一个车它能驾驶，作为一条狗它能叫。这些在对象中的功能在Java中被称为函数，所以我们可以使用这种命名方式（参见章节7.1）。调用对象立的函数需要通过点语法完成：

```
variableName.objectMethod(Method Arguments);
```

在车的例子中，我们的函数没有可用的参数选项，所以就看起来像这样：

```
myCar.draw();
myCar.display();
```

 练习8-3：假设有一人的类。你想要写一个代码声明人的对象以及使用函数sleep()。请写出下面的代码：

Declare and initialize the Human object: _____
Call the sleep() function: _____

8.5 用一个标签(tab)把他们放到一起

现在我们已经学习了如何定义一个类并且在类中使用一个对象，我们可以将在8.2章节和8.3章节所学到的放在一个程序中。

例 8-1：一个汽车类以及一个汽车对象

```
Car myCar;                                声明一个对象作为全局变量

void setup() {
    size(200,200);
    // Initialize Car object
    myCar = new Car();
}

void draw() {
    background(0);
    // Operate Car object
    myCar.move();
    myCar.display();
}
```

通过函数构造器在setup()中初始化汽车对象

通过使用点语法调用对象中的函数在draw()中使用汽车对象。

```

class Car {
    color c;
    float xpos;
    float ypos;
    float xspeed;
}

Car() {
    c = color(255);
    xpos = width/2;
    ypos = height/2;
    xspeed = 1;
}

void display() {
    // The car is just a square
    rectMode(CENTER);
    fill(c);
    rect(xpos,ypos,20,10);
}

void move() {
    xpos = xpos + xspeed;
    if (xpos > width) {
        xpos = 0;
    }
}

```

程序下面的其余部分用来定义一个类

变量（数据）

函数构造器

函数

函数

你会发现汽车类的代码块会放在主程序的下面(在draw()下面)。这点和在第7章中我们变成的方法上是相同的。从技术上讲，为了不混淆，只要代码块保持不变(包含大括号在内的)。汽车类应该可以纺织在setup()上面或者是setup()与draw()只见。虽然在技术上这是正确的，在编程时这种逻辑也是非常符合人类大脑的。但是，Processing中提供了一个非常好用的装置，通过使用标签 (tabs) 讲这些代码块分类。

在你的Processing窗口中，在右上角找到一个箭头在方形内部。如果你按下这个按钮你会看到它提供了一个新建标签 (New Tab)，如图8.1所示。

选择”新建标签”后你会看到提示输入新标签的名称，如图8.2所示。

虽然你可以随意选择你喜欢的名字，但是将你要编写的类的名称作为标签名可能是一个很实用的选择。然后你可以在一个标签中输入你的代码主体(在图8.2中名称为” objecteExample ”)，另一个标签中输入你的类(在图中名为” Car ”)。

在标签只见切换非常简单，只需要点击选项卡上的标签名称即可，如图8.3所示。此外，需要注意的是当你新建一个标签时，一个新的pde文件会在你的草图文件夹中创建，如图8.4。这个程序会有一个objectExample.pde文件和一个Car.pde文件。

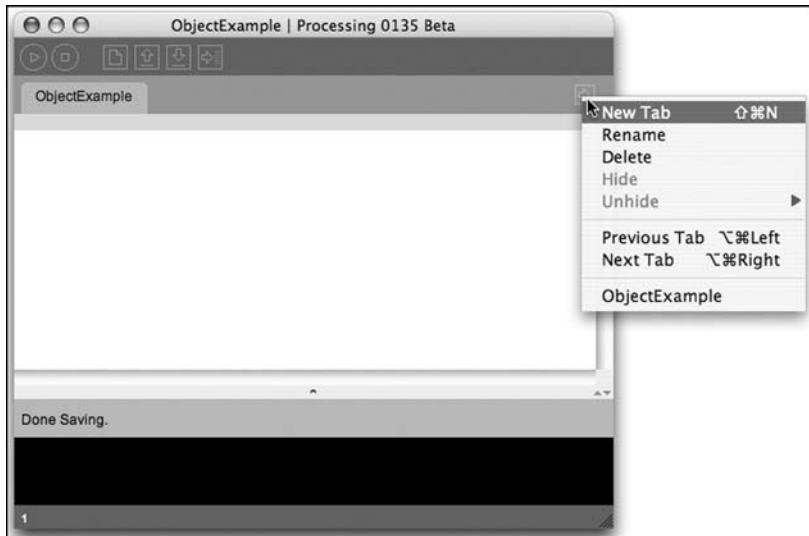


图 8.1

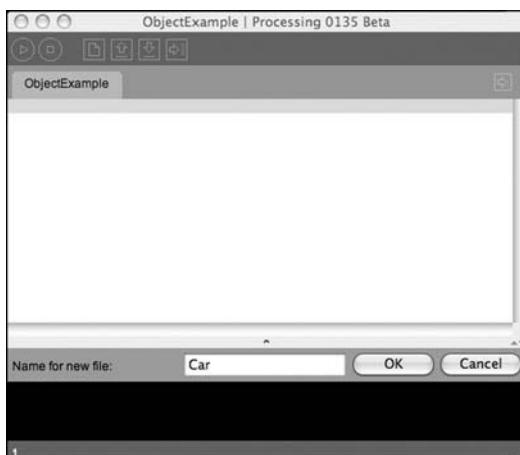


图 8.2

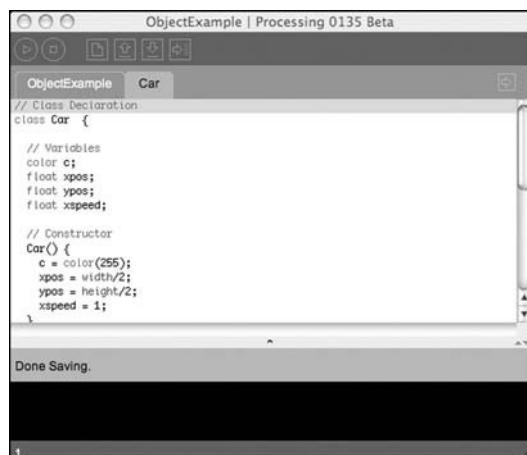


图 8.3



图 8.4

 练习8-3：创建一个多标签草图，尝试套用汽车的例子，不要有任何错误！

8.6 函数构造器的参数

在之前的例子中，对于汽车对象的初始化我们使用的都是像下面的new运算符。

```
Car myCar = new Car();
```

在我们学习OOP(面向对象编程)时这种写法是一种非常有用的简化。然而上面的代码却有一个相当严重的问题。当我们想在一个程序中写2辆车对象的时候该怎么写？

```
// Creating two car objects
Car myCar1 = new Car();
Car myCar2 = new Car();
```

这样就完成了我们想要的目标，这些代码能够写出2辆汽车的对象，一个变量存储在myCar1中，另一个变量存储在myCar2中。不过你研究汽车类，你会发现这2辆车是相同的：每一个都是白色，都从屏幕中间开始，并有一个1的速度。在口语表达中就好像：

创建一个新的车

而我们想要说的是：

创建一个新的红色汽车，初始位置在(0,10)，并有一个1的速度。

或者我们换另一种车：

创建一个新的蓝色汽车，初始位置在(0,100)，并有一个2的速度。

为此，我们需要将参数放在函数构造器内，实现不同的对象：

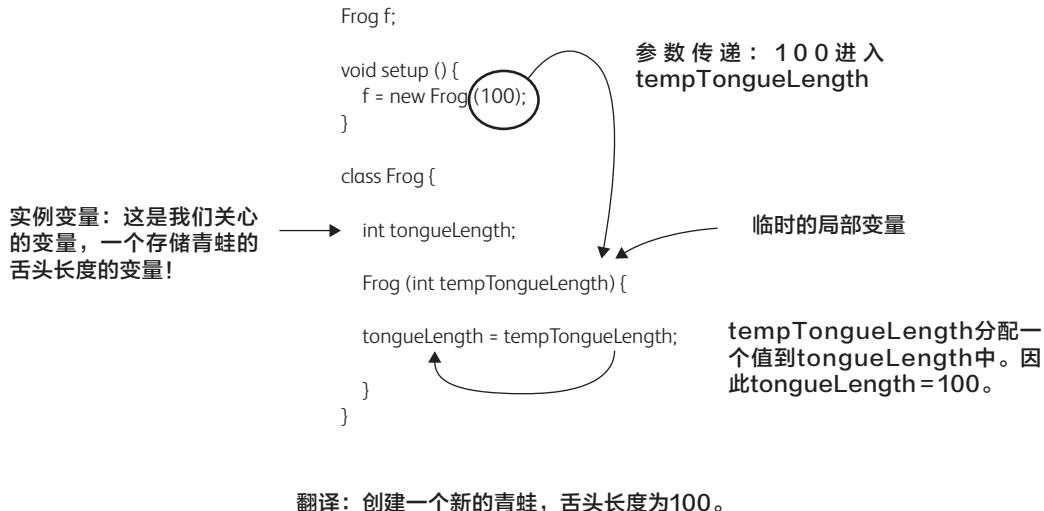
```
Car myCar = new Car(color(255,0,0),0,100,2);
```

函数构造器必须重新将这些参数合并起来。

```
Car(color tempC, float tempXpos, float tempYpos, float tempXspeed) {
    c = tempC;
    xpos = tempXpos;
    ypos = tempYpos;
    xspeed = tempXspeed;
}
```

在我的经验中，使用函数构造器中的参数初始化对象变量可能有些让人眼花缭乱。这不是你的错。这写代码看起来非常奇怪并且很累赘：对于每一个我需要初始化的变量，我必须在构造器里面写这些零时的参数吗？

然而，这确实是一个很重要的功能，并且它是面向对象编程中最强大的功能之一。但是现在，它可能回显得很杂乱。让我们简要回顾以下参数传递是如何在这些代码块中工作的。如图8.5所示。



参数是一个函数内部的局部变量，并且当你调用时你需要填充参数。在例子中，他们只有一个目的，在一个对象中初始化变量。这些变量就是汽车例子中的实际位置。函数构造器中的参数仅仅时一个临时的，并且只是作为对象本身传递一个值。

它允许我们在同样的函数构造器中写出各种各样不同的对象。你可能会将你的临时参数名字叫做tempC。在实际编程中你会看到有写程序员喜欢叫做c_。你可以命任何你想要的名字，当然需保持整体的统一有助于识别。

现在我们可以看看在一个程序中写出不同性质的对象了。

例 8-2：2个汽车对象

```

Car myCar1;
Car myCar2;

void setup() {
    size(200,200);
    myCar1 = new Car(color(255,0,0),0,100,2);
    myCar2 = new Car(color(0,0,255),0,10,1);
}

```

2个对象

```

void draw() {
    background(255
}

```

当对象有函数构造器，参数就在括号内

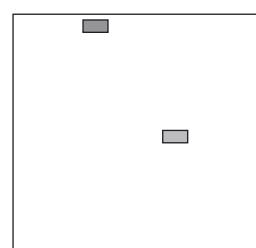


图 8.6

```

myCar1.move();
myCar1.display();
myCar2.move();
myCar2.display();
}

class Car {
    color c;
    float xpos;
    float ypos;
    float xspeed;

    Car(color tempC, float tempXpos, float tempYpos, float tempXspeed) {
        c = tempC;
        xpos = tempXpos;
        ypos = tempYpos;
        xspeed = tempXspeed;
    }

    void display() {
        stroke(0);
        fill(c);
        rectMode(CENTER);
        rect(xpos,ypos,20,10);
    }

    void move() {
        xpos = xpos + xspeed;
        if (xpos > width) {
            xpos = 0;
        }
    }
}

```

虽然这里有多个对象，但我们只需要一个类。不管有多少饼干要做，我们只需要一个饼干模型即可。面向对象编程是不是很激情？

函数构造器定义参数。



练习8-5：运用对象重写第5章中重力球类的例子。并且写出2个对象。原来的例子在这里为你提供一个框架帮助你开始，仅供参考。

```

_____._____:
Ball ball2;

float grav = 0.1;

void setup() {
    size(200,200);
    ball1 = new _____(50,0,16);
    _____(100,50,32);
}

```

```

void draw() {
    background(100);
    ball1.display();
    _____
    _____
}

_____  

float x;  

_____  

float speed;  

float w;  

_____  

_____(_____,_____,_____) {  

    x = ____;  

    _____  

    speed = 0;  

}  

void _____() {  

    _____
    _____
}
    _____
    _____
    _____
    _____
}

```

```

// Simple gravity
float x = 100;           // x location
float y = 0;             // y location

float speed = 0;          // speed
float gravity = 0.1;      // gravity

void setup() {
    size(200,200);
}

void draw() {
    background(100);

    // display the square
    fill(255);
    noStroke();
    rectMode(CENTER);
    rect(x,y,10,10);

    // Add speed to y location
    y = y + speed;

    // Add gravity to speed
    speed = speed + gravity;

    // If square reaches the bottom
    // Reverse speed
    if (y > height) {
        speed = speed * -0.95;
    }
}

```

8.7 对象也是数据类型

这是我们对于面向对象编程的第一次经验，所以我们很容易运用。本章中的例子只使用了一个类，针对这个类的对象最多2到3个。不过实际上这个也没有限制。Processing的草图能够多个包括任何你想写的类，例如如果你在编写“太空侵略者”游戏，你可以会创建太空船类、敌人类、子弹类，而用对象作为你游戏中的每一个实体。

此外，虽然不是原始类型数据，但是类也是数据类型，就像整数和浮点数一样。由于类是数据类型，所以一个对象可以包含其他对象。例如，假设你刚刚写完一个叉子和勺子的类的编程。移动到一个PlaceSetting类里面，那PlaceSetting类可能就会包括叉子对象和勺子对象。在面向对象编程中这是完全合理并且相当普遍的。

```
class PlaceSetting {
    Fork fork;
    Spoon spoon;
    PlaceSetting() {
        fork = new Fork();
        spoon = new Spoon();
    }
}
```

一个类可以包括其他对象的变量

对象就像其他数据类型一样，也可以作为参数传递给函数。在太空侵略者游戏例子中，如果飞船把子弹射向敌人，我们可能会想要写一个函数，以确定敌人类是否被子弹击中。

```
void hit(Bullet b) {
    // Code to determine if
    // the bullet struck the enemy
}
```

一个函数能够将对象作为它的参数

在第7章中，我们展示了如何通过一个函数中的原始值变成一个副本。在有对象时，是不会发生这种情况，并且结果会更直观。如果在函数中改变了对象后，这些变化会影响其他任何使用这个对象的地方。这被称为引用传递，而不是复制副本，它会传递给函数的实际对象本身的引用。

随着我们不断向前的学习，在这本书中的例子将变得越来越有难度，我们会看到例子使用好几个对象，通过对对象传递到函数中以及更多的东西。下一章，实际上就是学习如何为对象创建一个列表。第10章的项目会在一个项目中包含好几个类。现在，我们会离开Zoog，我们会坚持只使用一个类。

8.7 面向对象的Zoog

虽然步子在向前，但我们唯一没有变的是我们的问题又来了：我们应该在什么时候使用面向对象编程？对我来说，答案是始终使用。对象能让你组织一些模块化的东西导入一个软件应用中去，并且可以重复使用。你会看到这本书中会不断重复对象。然而它也并不是在所有的地方都很方便，尤其是在你学习的时候。Processing能够在没有对象代码的情况下很简单地写一个简易“草图”视觉化创意。

对于任何你想做的Processing项目，我的建议是采取一步一步的方法。你不需要一开始就写类让它能做所有的事。首先勾勒出你在setup()和draw()中要写的代码。你想要做什么，以及你想看到会发生什么。随着项目慢慢变复杂，再花时间去重新组织你的代码，也许是先用函数再用对象。在不改变最终结果的情况下花大量时间去重新组织代码（通常叫作重构）这种方法是完全可行的。

这也正是我们从第1章到现在所在做Zoog的这些事。我们勾勒出Zoog，并且尝试一些运动行为。现在我们已经确定了这些运动，我们就可以将Zoog重构进对象中了。这个过程会在我们应对更复杂的编程时助我们一臂之力。

所以现在是时候重新整理，做出Zoog的类。我们的小Zoog几乎都长大了。下面的例子和例7-5几乎相同(带有函数的Zoog)但是有一个很大的不同。在例7-5中所有的变量以及所有的函数现在都放入到Zoog的类中，所以setup()和draw()中几乎没有任何代码。

例 8-3：带有函数的Zoog

```
Zoog zoog; zoog是一个对象！

void setup() {
    size(200,200);
    smooth();
    zoog = new Zoog(100,125,60,60,16); 通过构造器初始化Zoog的属性。
}

void draw() {
    background(255);
    // mouseX position determines speed factor
    float factor = constrain(mouseX/10,0,5);
    zoog.jiggle(factor);
    zoog.display(); Zoog能用函数做些东西！
}

class Zoog {
    // Zoog's variables
    float x,y,w,h,eyeSize;

    // Zoog constructor
    Zoog(float tempX, float tempY, float tempW, float tempH, float tempEyeSize) {
        x = tempX;
        y = tempY;
        w = tempW;
        h = tempH;
        eyeSize = tempEyeSize;
    }
}
```

一切与Zoog有关的东西都包含在这个一个类中。Zoog有属性(位置，宽度，高度，眼睛等等)并且Zoog也有一些能力(摇摆，显示)。

```

// Move Zoog
void jiggle(float speed) {
    // Change the location of Zoog randomly
    x = x + random(-1,1)*speed;
    y = y + random(-1,1)*speed;

    // Constrain Zoog to window
    x = constrain(x,0,width);
    y = constrain(y,0,height);
}

// Display Zoog
void display() {
    // Set ellipses and rects to CENTER mode
    ellipseMode(CENTER);
    rectMode(CENTER);

    // Draw Zoog's arms with a for loop
    for (float i = y - h/3; i < y + h/2; i += 10) {
        stroke(0);
        line(x-w/4,i,x + w/4,i);
    }

    // Draw Zoog's body
    stroke(0);
    fill(175);
    rect(x,y,w/6,h);

    // Draw Zoog's head
    stroke(0);
    fill(255);
    ellipse(x,y-h,w,h);

    // Draw Zoog's eyes
    fill(0);
    ellipse(x-w/3,y-h,eyeSize,eyeSize*2);
    ellipse(x + w/3,y - h,eyeSize,eyeSize*2);

    // Draw Zoog's legs
    stroke(0);
    line(x - w/12,y + h/2,x - w/4,y + h/2 + 10);
    line(x + w/12,y + h/2,x + w/4,y + h/2 + 10);
}

```

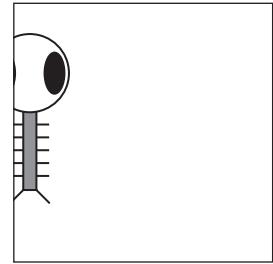


图 8.7



练习8-5：重写例8-3，让它包含2个Zoog。你可以改变他们的外表以及行为，或者考虑为Zoog添加一个色彩变量。



第二课项目

- 步骤1** 将将第二课的项目使用函数的方式写出来并且重组。
- 步骤2** 使用类和对象变量进一步重组代码。
- 步骤3** 在你创建的类中的函数构造器添加参数，并尝试作出2到3个不同的对象。

第四课

更多相同的

9.数组

9. 数组

“我可能会慢慢的不断的重复自己，一个美丽心灵的报价单 – 如果我能够记住任何东西。”

— Dorothy Parker

本章内容：

- 什么是数组
- 声明一个数组
- 初始化
- 数组操作 – 在数组中使用”for”循环
- 对象中的数组

9.1 数组，为什么我们需要它

让我们花点时间重温以下前一章面向对象编程的例子。你可能还记得我们话了很大的精力去做一个包含一个类和2个对象的实例。

```
Car myCar1;  
Car myCar2;
```

作为电脑程序员的生活中，这的确算是一个激动人心的时刻。也许你正在考虑一个比较明显的问题，你怎么能够让程序更升一级，让它有100个对象？一些聪明的人会复制粘贴，可能就像下面一样：

```
Car myCar1  
Car myCar2  
Car myCar3  
Car myCar4  
Car myCar5  
Car myCar6  
Car myCar7  
Car myCar8  
Car myCar9  
Car myCar10  
Car myCar11  
Car myCar12  
Car myCar13  
Car myCar14  
Car myCar15  
Car myCar16  
Car myCar17  
Car myCar18  
Car myCar19  
Car myCar20  
Car myCar21
```

Car myCar22
Car myCar23
Car myCar24
Car myCar25
Car myCar26
Car myCar27
Car myCar28
Car myCar29
Car myCar30
Car myCar31
Car myCar32
Car myCar33
Car myCar34
Car myCar35
Car myCar36
Car myCar37
Car myCar38
Car myCar39
Car myCar40
Car myCar41
Car myCar42
Car myCar43
Car myCar44
Car myCar45
Car myCar46
Car myCar47
Car myCar48
Car myCar49
Car myCar50
Car myCar51
Car myCar52
Car myCar53
Car myCar54
Car myCar55
Car myCar56
Car myCar57
Car myCar58
Car myCar59
Car myCar60
Car myCar61
Car myCar62
Car myCar63
Car myCar64
Car myCar65
Car myCar66
Car myCar67
Car myCar68
Car myCar69
Car myCar70
Car myCar71
Car myCar72
Car myCar73
Car myCar74
Car myCar75
Car myCar76
Car myCar77
Car myCar78
Car myCar79

```
Car myCar80  
Car myCar81  
Car myCar82  
Car myCar83  
Car myCar84  
Car myCar85  
Car myCar86  
Car myCar87  
Car myCar88  
Car myCar89  
Car myCar90  
Car myCar91  
Car myCar92  
Car myCar93  
Car myCar94  
Car myCar95  
Car myCar96  
Car myCar97  
Car myCar98  
Car myCar99  
Car myCar100
```

如果你按照上面的方法完成这个编程你肯定会很头痛。这是一个很~~2~~的行为。

一个数组就能够将这100行代码放入到一行代码中，并且不需要100个变量。

在任何编程中都需要多个相似数据的实例，这就是该使用数组的时候了。例如，一个数组能够存储一场球赛中4个球员的分数，一个编程中选择的10个颜色，或者是水族馆中一系列鱼的对象。

 练习9-1：看看你到现在为止所创建的所有草图，是不是都没有使用到数组？为什么？

9.2 什么是数组

从第4章中，你可能还记得函数命名是指向数据储存中的一个点。换句话说，变量使得应用程序能够在一段时间内追踪信息。一个数组的概念是完全一样的，唯一不同的是它指向的不是单个信息点，一个数组会指向多个信息点，如图9.1所示。

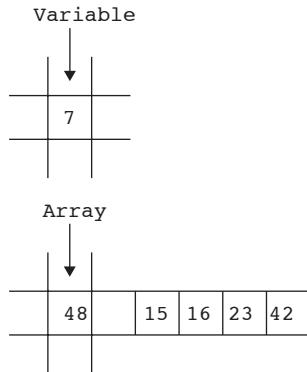


图 9.1

你可以把数组当作一组变量。一个组，但应该注意，在2种情况下是非常有用的。1,追中自身组里的元素。2,追踪列表中这些元素的顺序（这些元素可能是第一个，第二个，第三个等等）。在很多编程中这是非常关键的，因为在很多程序中信息的顺序和信息本身同样重要。

在一个数组中，每一个元素都有它自身的索引，它会用一个整数指定在数组中（如element #1, element #2等等）。在所有情况下，该数组的名称会引用到整个数组中，每一个元素能够通过名称访问它的位置。

请注意在图9.2中，索引的范围从0到9。这个数组有合计10个元素，第一个元素是0,最后一个元素是9。我们可能会开始抱怨：“嘿，为什么我们的编号不是从1到10？那样不是会更容易吗？”

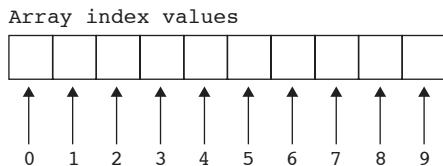


图 9.2

尽管是第一位，我们直观的感觉应开从1开始计数（有一些编程语言是这样），我们从0开始计数是应该在技术上数组的第一个元素是数组的开端，距离从0开始。从0开始编号的元素也会使许多数组操作（执行一行代码列表中的每一个元素）更方便。通过我们接下来的几个例子，你会开始相信从0开始计数确实很实用。



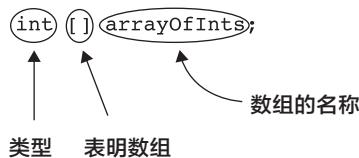
练习9-2：如果你有一个包含1000个元素的数组，索引值的范围应该是从哪到哪？

答案：从_____ 到 _____

9.3 声明和创建数组

在第4章中，我们了解到所有的变量都必须有一个名称以及一个数据类型。数组也一样。但是声明数组的时候却不同。我们会在声明数组类型之后加上一个方括号([])。让我们以原始值为例，如整数。(数组可以声明任何类型的数据，这里我们将看到我们怎么创建一个对象数组)如图9.3所示。

图 9.3



在图9.3的声明中表示“arrayOfInts”会存储一组整数。这个数组的名称叫做“arrayOfInts”可以是任何你想要的东西(我们这里只包含单词“array”说明我们正在学这个)。

数组有一个基本属性，那就是它们的大小是固定的。一旦我们定义了数组的大小，它就永远不会改变。一个10个整数的数组永远不能改成11个整数。但是上面的代码在哪里定义了数组的大小？上面的代码仅仅声明了数组，我们还需要创建一个实例指定数组的大小。

要定义这些，我们需要使用new操作符，就像我们调用一个对象的方式类似。在对象中，我们说“创建一个新车”或者“创建一个新Zoog”。而在数组中我们会说“创建一个新的整数数组”或者“创建一个新的汽车对象数组”等等。如图9.4所示。

图 9.4

数组的声明和定义



在图9.4中数组的声明允许我们指定数组的大小：我们希望有多少个元素在数组上保存(或者从技术上说，我们希望有多少的内存数组需要存储)。我们写的语句如下：new操作符，后面跟着数据类型，后面跟着方括号(方括号里面是数组的大小)。这个大小必须是一个整数，它可以是一个普通的数字，或者一个变量(整数类型的)，或者是一个结果为整数的算式表达(如2+2)。

例 9-1：其他数组声明及创建的例子

```

float [] scores = new float [4];           //一组4个浮点数的数组
Human [] people = new Human [100];        //一组100个人类对象的数组
int num = 50;
Car [] cars = new Car [num];              //用一个变量指定大小
Spaceship [] ships = new Spaceship [num*2+3]; //用一个表达式指定数组大小

```



练习9-3：写出下列数组的声明语句

30个整数

100个浮点数

56个Zoog对象



练习9-4：下列数组哪写是有效的那些是无效的，为什么？

int[] numbers = new int[10];

float[] numbers = new float[5 + 6];

int num = 5;
float[] numbers = new int[num];

float num = 5.2;
Car[] cars = new Car[num];

int num = (5 * 6)/2;
float[] numbers = new
float[num = 5];

int num = 5;
Zoog[] zoogs = new Zoog[num * 10];

情况正在好转，我们不仅成功的声明了一个数组，并且给予它大小以及分配储存数据。但是一个主要的东西还没有：那些需要被储存的数据本身。

9.4 初始化数组

一种方法是用硬代码填充数组中的每一个数据点。

例 9-2：一个一个的初始化数组

```
int [] stuff = new int[3];
stuff [0] = 0;           //数组中的第一个元素是0
stuff [1] = 3;           //数组中的第二个元素是3
stuff [2] = 1;           //数组中的第三个元素是1
```

正如你看到的，我们给数组中每一个元素单独指定一个索引，从0开始。数组的语法就是数组的名称后面接着括号，括号中是索引值。

arrayName[INDEX]

第二种方法就是用大括号，在大括号中输入每一个元素的数值，中间用逗号隔开。

例 9-3：一次性初始化数组

```
int [] arrayOfInts = {1,5,8,9,4,5};
float [] floatArray = {1.2,3.5,2.0,3.4,123,9.9};
```



练习9-5：声明一个包含3个Zoog对象的数组。通过它们的索引初始化数组中的每一个点。

```
Zoog__ zoogs = new _____ [_____];
_____ [_____ ] = _____ _____(100, 100, 50, 60, 16);
_____ [_____ ] = _____ _____(_____ );
_____ [_____ ] = _____ _____(_____ );
```

这两种方法在并不经常使用，并且在这本书中的大多数例子不会出现。实际上，初始化的方法在本章的开头提出的那个问题。想象一下一个一个的初始化100个元素，1000个元素甚至更多（恐怖）。

所有的这些问题，最终会有一个解决方案。叮叮叮，一个响亮的点子在脑中响起。循环！（如果你忘记了，去重温第6章）

9.5 数组操作

考虑一下下面的问题：

- (A) 创建一个包含1000个浮点数的数组。(B) 初始化它们每一个元素让每一个元素指定为0到10中的一个随机数。

A部分我们已经知道该怎么做了。

```
float [] values = new float [1000];
```

我们现在想避免的就是B部分：

```
values[0] = random(0,10);
values[1] = random(0,10);
values[2] = random(0,10);
values[3] = random(0,10);
values[4] = random(0,10);
values[5] = random(0,10);
等等
```

让我们用口语描述我们想要的程序：

对于每一个从0到99的数字n，初始化的值为0到10之间。翻译成代码，我们便有了：

```
int n=0;
values[n] = random(0,10);
values[n+1] = random(0,10);
values[n+2] = random(0,10);
values[n+3] = random(0,10);
values[n+4] = random(0,10);
values[n+5] = random(0,10);
```

尽管我们已经迈出了一大步，但是不幸的是情况并没有好转。现在使用一个变量(n)描述一个数组中的索引，我们可以采用一个while循环初始化每一个元素n。

例 9-4：使用 while 循环初始化数组中的所有元素

```
int n = 0;

while(n<1000){
    value [n] =random (0,10);
    n = n+1;
}
```

使用for循环能够让我们做的更简洁，如例9-5所示。

例 9-5：使用 for 循环初始化数组中的所有元素

```
for(int n = 0, n < 1000, 0++){
    values[n] = random(0,10);
}
```

我们可以运用相同的方式对应任何数组操作，我们可以做的事远远超出了简单的元素初始化。如，我们可以创建一个数组，每个元素之间的值事前一个的2倍(我们现在将使用i代替n，因为通常程序员都这么写)。

例 9-6：数组操作

```
for (int i = 0; i < 1000; i++) {
    values[i] = values[i] * 2;
}
```

在例9-6中有一个问题：使用硬代码值1000次，努力成为更好的程序员，我们应该始终对硬代码提出质疑。在这种情况下，如果我们想改变数组让他编程2000个元素怎么办？如果我们有很长很长的数组操作，我们不得不在整块代码中改变我们的代码。幸运的事，Processing给予我们一个很好的访问数组大小的手段，使用我们在第8章中所学到的点语法。长度事每一个数组的属性，并且我们访问它的时候可以这么说：

arrayName dot length

我们用长度清除数组。它将让每一个值重置为0。

例 9-7：使用.length数组操作

```
for (int i = 0; i < values.length; i++) {
    values[i] = 0;
}
```



练习 9-5：假设一个包含 10 个整数的数组

int[] nums = {5,4,2,7,6,8,5,2,8,14};

编写下面代码来执行下面的数组操作(注：每个数字的索引不同，因为[____]写的不明确，但不代表去掉括号)。

每个元素的值都是 它们索引号码的平 方	for (int i ____; i < ____; i + +) { ____[i] = _____ * _____; }
---------------------------	---

在每个索引之间添加一个0到10之间的随机数。	for (int i ____; i < ____; i + +) { ____[i] = _____ * _____; }
在数组中每个索引之间添加数字，跳过数组中的最后一个值。	for (int i = 0; i < ____; i++) { _____ += _____ [____]; }
计算所有数字的总和。	_____ = ____; for (int i = 0; i < nums.length; i++) { _____ += _____; }

9.6 简单的数组实例：蛇

让编程跟随鼠标移动，这看起来是一个非常简单的任务，但是在最初可能比较难。解决方案是运用一个数组，它能够存储鼠标的坐标的历史。我们将使用2个数组，一个数组用于存储横向坐标信息，另一个用于存储纵向坐标信息。比方说我们想存储鼠标最近的50个坐标。

首先我们要声明2个数组。

```
int[] xpos = new int[50];
int[] ypos = new int[50];
```

第二步，我们必须在setup()中初始化数组。由于编程开始没有任何鼠标移动，我们只需要将其填充为0。

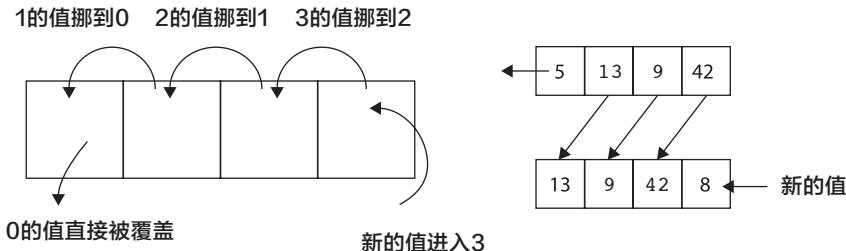
```
for (int i = 0; i < xpos.length; i++) {
    xpos[i] = 0;
    ypos[i] = 0;
}
```

每一次通过 draw()主循环，我们就会向数组更新当前鼠标的坐标。让我们选择将当前鼠标位置放置在数组中最后的一个位置中。数组的长度是50,这意味着索引值的范围在0 - 49.之间。最后一个索引值的位置是49,或者说这是数组的长度减1。

```
xpos[xpos.length-1] = mouseX;
ypos[ypos.length-1] = mouseY;
```

在数组中的最后一个索引数应该是数组的长度减1

现在最困难的一部分到了。我们只希望保留最近的50个鼠标坐标通过存储当前鼠标坐标在数组的末尾，我们要覆盖先前存储的信息。如果鼠标的第一帧在(10,10)。第二帧在(15,15)，我们想让(10,10)自动挪到数组中倒数第二个位置，而(15,15)变成最后一个位置。解决的方法是在更新当前坐标之前将所有的数组元素向下挪一个点。如图9.5所示。



元素索引49的值移动到48, 48的值移动到47, 47移动到46,以此类推。我们可以通过一个循环，并设置每个元素的索引i到元素加上1。注意我们必须停止到倒数第二个值，因为只有元素49没有元素50.换句话说，让它有一个条件。

```
i < xpos.length;
```

我们应该这样代替它：

```
i < xpos.length - 1;
```

完整的移动数组应该如下所示：

```
for (int i = 0; i < xpos.length - 1; i++) {
    xpos[i] = xpos[i + 1];
    ypos[i] = ypos[i + 1];
}
```

最终，我们可以使用鼠标坐标的历史绘制一系列的圆。每一个x, y数组的元素能够在相应存储的数组中绘制一个圆。

```
for (int i = 0; i < xpos.length; i++) {
    noStroke();
    fill(255);
    ellipse(xpos[i], ypos[i], 32, 32);
}
```

我们可以让这个更有意思一些，我们可以将圆的亮度以及大小与圆的坐标形成相关联系，先前的坐标会越来越亮越来越小，后来（新的）坐标会很暗，并且很大。这可以通过使用计数变量i来结合颜色以及尺寸。

```

for (int i = 0; i < xpos.length; i++) {
    noStroke();
    fill(255 - i*5);
    ellipse(xpos[i],ypos[i],i,i);
}

```

把上面这些步骤放在一起就形成了我们下面的例子，如图9.6所示。

例 9-8：一条跟着鼠标的蛇

```

// x and y positions
int[] xpos = new int[50];
int[] ypos = new int[50];

void setup() {
    size(200,200);
    smooth();

    // Initialize
    for (int i = 0; i < xpos.length; i++){
        xpos[i] = 0;
        ypos[i] = 0;
    }
}

void draw() {
    background(255);

    // Shift array values
    for (int i = 0; i < xpos.length-1; i++){
        xpos[i] = xpos[i + 1];
        ypos[i] = ypos[i + 1];
    }

    // New location
    xpos[xpos.length-1] = mouseX;
    ypos[ypos.length-1] = mouseY;

    // Draw everything
    for (int i = 0; i < xpos.length; i++){
        noStroke();
        fill(255-i*5);
        ellipse(xpos[i],ypos[i],i,i);
    }
}

```

声明2个包含50个元素索引的数组

初始化每一个元素的值，让它等于0

转移所有元素，让它代替下一个点。xpos[0] = xpos[1], xpos[1] = xpos[2],以此类推。在倒数第二个元素的时候停止。

在数组中最后一个索引更新鼠标的当前的坐标。

在每一个数组元素中都画一个圆。颜色以及尺寸都取决于循环计数器中的i的值。



图 9.6



练习9-7：利用面向对象变成重写蛇的例子，让蛇有一个类。让蛇看起来略有不同（不同的形状，颜色，大小）（还有一个高级的问题，创建一个点的类，让它能存储x和y的坐标。每一个蛇对象都有一个点对象数组，而不是像之前那样x和y是独立的值。涉及到对象数组的相关内容在下一节中学习）。

9.7 对象数组

我知道我知道，我还没有完全回答这个问题，我们如何写一个100个汽车对象的编程。

结合面向对象编程与数组能够实现让1个对象到10个对象甚至到1000个对象的简单转化。实际上，如果我们仔细点，我们就会发现不用去改变汽车的类。类与对象的多少没有一点关系。所以假设我们保持汽车的类的代码是相同的，让我们来看看如何使用对象数组。

让我们重温以下一个汽车对象的编程：

```
Car myCar;
void setup() {
    myCar = new Car(color(255,0,0),0,100,2);
}
void draw() {
    background(255);
    myCar.move();
    myCar.display();
}
```

上面代码中有3个步骤，我们需要利用数组改变每一个步骤。

之前

声明汽车对象

```
Car myCar;
```

初始化汽车对象

```
myCar = new Car(color(255),0,100,2);
```

通过调用的方式使用汽车对象

```
myCar.move();
myCar.display();
```

现在

声明汽车对象

```
Car[] cars = new Car[100];
```

初始化汽车对象数组

```
for (int i = 0; i < cars.length; i++) {
    cars[i] = new Car(color(i*2),0,i*2,i);
}
```

通过调用的方式使用汽车对象数组

```
for (int i = 0; i < cars.length; i++) {
    cars[i].move();
    cars[i].display();
}
```

这给我们留下了例9-9。注意如何改变汽车的数量，在数组中只需要改变在一开始定义的数组数量即可。其他人和地方都不能改变汽车的数量。

例 9-9：汽车对象的数组

```
Car[] cars = new Car[100];
```

一个包含100个汽车的数组

```
void setup() {
    size(200,200);
    smooth();
    for (int i = 0; i < cars.length; i++) {
        cars[i] = new Car(color(i*2),0,i*2,i/20.0);
    }
}
```

用for循环初始化每一个汽车

```
void draw() {
    background(255);
    for (int i = 0; i < cars.length; i++) {
        cars[i].move();
        cars[i].display();
    }
}
```

用for循环运行每一个汽车

```
class Car {
    color c;
    float xpos;
    float ypos;
    float xspeed;
```

```
Car(color c_, float xpos_, float ypos_, float xspeed_) {
    c = c_;
    xpos = xpos_;
    ypos = ypos_;
    xspeed = xspeed_;
}
```

```
void display() {
    rectMode(CENTER);
    stroke(0);
    fill(c);
    rect(xpos,ypos,20,10);
}
```

```
void move() {
    xpos = xpos + xspeed;
    if (xpos > width) {
        xpos = 0;
    }
}
```

汽车的类完全不用改变，就可以作出100个甚至1000个汽车

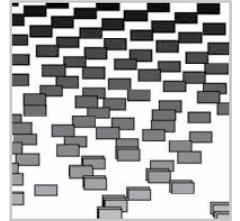


图 9.7

9.8 交互式对象

当我们第一次了解到变量(第4章)和条件语句(第5章)的时候，我们编写了一个很简单的翻转效果。一个矩形在窗口中显示，当鼠标在顶部时时一种颜色，当鼠标不在顶部时又是另一种颜色。下面是一个一个采用这种方式的简单例子，并且结合了”条纹”对象，虽然有10个条纹，但是每一个都有鼠标单独的翻滚函数rollover()。

```
void rollover(int mx, int my) {
    if (mx > x && mx < x + w) {
        mouse = true; } else {
        mouse = false;
    }
}
```

这个函数是检查是否有一个点(mx, my)是否在垂直条纹上。这个点是否大于条纹的左边缘并且小于它的右边缘？如果是，布尔变量”mouse”就是真。当你设计你的类的时候，使用布尔变量去跟踪对象的属性会很方便，它就类似于一个开关。例如，一个汽车对象可以运行或者不运行，Zoog可以开心或者不开心。

布尔变量在条件语句里面使用，条纹对象中的display()函数用来显示条纹的颜色。

```
void display() {
    if (mouse) {
        fill(255);
    } else { fill(255,100);
    }
    noStroke();
    rect(x,0,w,height);
}
```

当我们在对象上调用rollover()函数的时候，我们可以通过使用mouseX和mouseY作为其参数。

```
stripes[i].rollover(mouseX,mouseY);
```

虽然我们可以直接通过mouseX和mouseY直接放在rollover ()函数中，但是最好的方法是把他们当作参数。这样就会有更大的灵活性。条纹对象能够检查以及确定是否有x, y坐标在矩形中。

下面是”交互条纹”的完整例子

例 9-10：交互条纹

```

// An array of stripes
Stripe[] stripes = new Stripe[10];

void setup() {
    size(200,200);

    // Initialize all "stripes"
    for (int i = 0; i < stripes.length; i++) {
        stripes[i] = new Stripe();
    }
}

void draw() {
    background(100);

    // Move and display all "stripes"
    for (int i = 0; i < stripes.length; i++) {

        // Check if mouse is over the Stripe
        stripes[i].rollover(mouseX,mouseY);
        stripes[i].move();
        stripes[i].display();
    }
}

class Stripe {
    float x; // horizontal location of stripe
    float speed; // speed of stripe
    float w; // width of stripe
    boolean mouse; // state of stripe (mouse is over or not?)

    Stripe() {
        x = 0; // All stripes start at 0
        speed = random(1); // All stripes have a random positive speed
        w = random(10,30);
        mouse = false;
    }

    // Draw stripe
    void display() {
        if (mouse) {
            fill(255);
        } else {
            fill(255,100);
        }

        noStroke();
        rect(x,0,w,height);
    }
}

// Move stripe
void move() {
    x+=speed;
    if (x > width + 20) x = -20;
}

```

通过鼠标坐标，转换到对象

一个布尔变量追
踪对象状态

布尔变量确定条纹的颜色

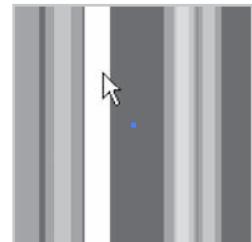


图 9.8

```
// Check if point is inside of Stripe  
void rollover(int mx, int my) {  
  
    // Left edge is x, Right edge is x + w  
    if (mx > x && mx < x+w) {  
        mouse = true;  
    } else {  
        mouse = false;  
    }  
}
```

检查点(mx,my)是否在条纹中

练习9-8：编写一个按钮的类（详见例5-5：一个面向对象按钮）。按钮的类应该是一个登陆器，当鼠标按倒按钮上的时候它会变色。再利用数组创建不同尺寸和坐标的按钮对象。在写主编程之前，先勾画出按钮的类。假设按钮一开始是关闭的。下面是代码框架：



```
class Button {  
    float x;  
    float y;  
    float w;  
    float h;  
    boolean on;  
    Button(float tempX, float tempY, float tempW, float tempH) {  
        x = tempX;  
        y = tempY;  
        w = tempW;  
        h = tempH;  
        on = _____;  
    }
```

9.9 Processing 的数组函数

OK,所以我制造可一个秘密。我撒谎了。在本章前面我非常重点的强调一旦设定了数组的大小，你就永远无法改变这种大小。一旦你创建了10个按钮对象，你就不能创建第11个。

并且我坚持那句话。从技术上说，当你给一个数组分配10个点，就表明你已经告诉Processing你打算使用多少内存空间给这个数组。你不能指望内存能够自动分配更多的空间给它，所以你不能扩大你数组的大小。

但是，为什么你不能只做一个新的数组（让它包含11个索引），从你之前的数组复制前10个，在最后一个索引加上一个新的按钮对象。实际上,在Porcessing中提供了一系列数组函数能够帮助你管理数组的大小。他们是：shorten(), concat(), subset(), append(), splice(), and expand()。此外，还有些函数能够改变数组里面的顺序，如sort() 和 reverse()。

所有这些函数的详细信息你可以在reference中找到。让我们来看看一个例子，它使用了append()函数来扩大数组的大小。这个例子(其中就包括了练习8-5的答案)以包含一个对象的数组开始。每一次按下鼠标，一个新的对象就会被创建，并且会追加到原来数组的末尾。

例 9-10：使用 append() 改变数组大小

```
Ball[] balls = new Ball[1];
float gravity = 0.1;
```

我们在最开始的数组中
只设定了一个元素

```
void setup() {
  size(200,200);
  smooth();
  frameRate(30);
```

```
// Initialize ball index 0
balls[0] = new Ball(50,0,16);
}
```

无论数组有多少对象，更
新并显示所有的对象。

```
void draw() {
  background(100);
```

```
// Update and display all balls
for (int i = 0; i < balls.length; i++) {
```

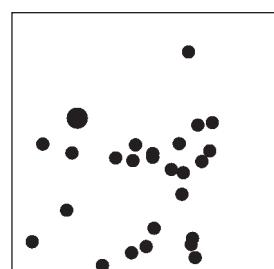


图 9.9

```

balls[i].gravity();
balls[i].move();
balls[i].display();
}

}

void mousePressed() {
// A new ball object
Ball b = new Ball(mouseX, mouseY, 10);

// Append to array
balls = (Ball[]) append(balls, b);
}

class Ball {
float x;
float y;
float speed;
float w;

Ball(float tempX, float tempY, float tempW)
x = tempX;
y = tempY;
w = tempW;
speed = 0;
}

void gravity() {

// Add gravity to speed
speed = speed + gravity;
}

void move() {

// Add speed to y location
y = y + speed;

// If square reaches the bottom
// Reverse speed
if (y > height) {
speed = speed * -0.95;
y = height;
}
}

void display() {

// Display the circle
fill(255);
noStroke();
ellipse(x, y, w, w);
}
}

```

在鼠标点击的位置创建一个新的对象

在这里，函数append()在数组的末尾追加了一个元素。append()要使用两个参数，第一个是数组中你需要追加的数组，第二个就是你想在那个数组中追加东西。你就会看到在原有数组上重新调整的结果。此外，append()函数需要你明确声明数据的类型，并将它放入数组括号前面：(Ball[])。这就是所谓的追加。

调整数组大小的方法不只这一个，还有一个就是使用一个特殊的对象，它被称为ArrayList，我们将在第23章中接触到。

9.9 一千零一个 Zoog

现在是时候完整Zoog的旅程了，让我们看看如何从一个Zoog对象变成多个。这里我们运用了汽车数组的例子，我们可以简单的复制例8-3的类，然后来进行数组。

例 9-10：使用append()改变数组大小

```
Zoog[] zoogies = new Zoog[200]; // 在这个例子中于之前唯一的区别是它使用了数组中包含多个Zoog。
void setup() {
    size(400,400);
    smooth();
    for (int i = 0; i < zoogies.length; i++) {
        zoogies[i] = new Zoog(random(width),random(height),30,30,8);
    }
}
void draw() {
    background(255); // Draw a black background
    for (int i = 0; i < zoogies.length; i++) {
        zoogies[i].display();
        zoogies[i].jiggle();
    }
}

class Zoog {
    // Zoog's variables
    float x,y,w,h,eyeSize;
    // Zoog constructor
    Zoog(float tempX, float tempY, float tempW, float tempH, float tempEyeSize) {
        x = tempX;
        y = tempY;
        w = tempW;
        h = tempH;
        eyeSize = tempEyeSize;
    }
    // Move Zoog
    void jiggle() {
        // Change the location
        x=x+random(-1,1);
        y=y+random(-1,1);
        // Constrain Zoog to window
        x = constrain(x,0,width);
        y = constrain(y,0,height);
    }
    // Display Zoog
    void display() {
        // Set ellipses and rects to CENTER mode
    }
}
```

为了简单起见，我们在 `jiggle()` 中还去掉了速度参数。尝试把加入速度当作练习。

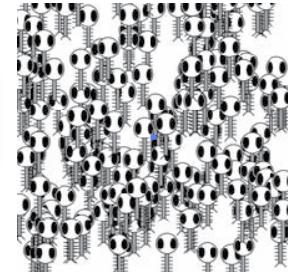


图 9.10

```
ellipseMode(CENTER);
rectMode(CENTER);

// Draw Zoog's arms with a for loop
for (float i = y-h/3; i < y + h/2; i+= 10) {
  stroke(0);
  line(x-w/4,i,x +w/4,i);
}

// Draw Zoog's body
stroke(0);
fill(175);
rect(x,y,w/6,h);

// Draw Zoog's head
stroke(0);
fill(255);
ellipse(x,y-h,w,h);

// Draw Zoog's eyes
fill(0);
ellipse(x-w/3,y-h,eyeSize,eyeSize*2);
ellipse(x+ w/3,y-h,eyeSize,eyeSize*2);

// Draw Zoog's legs
stroke(0);
line(x-w/12,y+ h/2,x-w/4,y+ h/2+ 10);
line(x+ w/12,y+h/2,x+ w/4,y+ h/2+10);
}
```



第四课项目

步骤1 使用在第三课中所学的类，利用类创建一个含有对象的数组。

步骤2 你能作出对象反应鼠标吗（互动）？尝试使用dist()函数来确定对象接近鼠标的位置。
例如，你可以使每一个接近鼠标的对象开始摇动。

多少个对象会让你的草图比之前的运行的速度慢？

使用下面的提供的空间为你的项目写一些草图。

第五课

整合所有部分

10. 算法

11. 调试

12. 库

10. 算法

“打泡沫，冲洗，重复。”

— 未知

10.1 我们在哪？我们要去什么地方？

我们的朋友Zoog已经有一个很好的运行。Zoog教会了我们Processing中记本的形状库的内容。从中Zoog可以运用鼠标进行有意思的交互，通过变量自动的移动，并且可以通过条件语句改变方向，或者是扩大它自身的循环，用函数将他们组织起来，将数据和函数封装到一个对象中去，最后利用数组复制自身。对于我们来说，这是一个很好的学习步骤。但是读完这本书之后你会发现将外来生物在屏幕周围运动的项目是几乎没有的。我们需要停下来想一想我们已经学到了什么，并且我们如何将学到的转化成去实践。你现在想要做的编程是什么，想一想怎么用变量、条件语句、循环、函数、对象以及数组来帮助你作出你想要的编程？

在之前的章节，我们编程的例子主要集中在单一的功能编程。Zoog能摇动但只会摇动。Zoog不能突然的跳起来，并且Zoog通常是一个，从来没有与其他外星生物进行什么有趣的交互。当然，我们可以使用一些早期的例子，但是重要的是基于例子中的那些基本功能我们能真正的学习到编程的基础。

在实际项目中，经常会涉及到许多移动部件。本章的目的就是为了展示如何在一个较大的项目中创建出一些较小的独立编程。你，程序员，需要用一个整体的视角来看待项目，但是必须学会如何把整体分解成单个的部分来执行这些项目。

我们会从一个想法开始，在阅读完这本书之后你可能会挑选一个“想法”的例子，基于这个例子作出项目，但是可悲的是没有。编写你自己的编程是一件非常令人激动的事，因为数组所带来的创作的可能性是不可估量的。最终，你将作出完全属于自己的东西。然而，现在我们需要挑一个简单的相马去学习基本只是，这不是一个实际的项目，我们能够插入一些编程中通用的东西，这样有助于我们在开发大型项目中学习。

我们选择的是一个简单的交互游戏的例子，多个对象，以及一个目标。聚焦虽然不是一个好的游戏设计，但可能会是一个好的软件设计。你将怎么把想法转换成代码呢？如何利用你自己的算法实现你的想法呢？我们将看到一个大项目是如何划分成4个小项目并一步一步完成的，最终把他们所有部分结合起来执行编程原来的想法。

我们将继续强调面向对象编程的重要性，每一个部分都会成为一个可用的类。当这些类齐全之后你会看到自给自足是多么容易，然后是创建最终的方案。让我们在执行想法以及分部之前，让我们回顾一个算法的概念，我们将在2a以及2b的步骤中需要这些概念。

1. **想法**—以想法为开端
2. **分解部件**—将想法分解成一些较小的部分
 - a. **伪代码算法**—将每一部分使用成伪代码的形式表现出来
 - b. **代码算法**—将伪代码转换成实际代码
 - c. **对象**—把相关的数据以及函数结合成算法，整合成一个类
3. **一体化**—结合第二步中的所有的类，将他们整合成一个大的算法

10.2 算法：舞起你自己的节拍

算法是用于解决问题的步骤或公式。在电脑编程中，算法是执行任务所需步骤的顺序。我们已经创建了迄今为止在这本书中涉及算法的例子。

算法就像做菜的方法一样：

1. 把烤箱预热到400° F
2. 把4个已经去骨的鸡胸肉放在烤盘上
3. 把芥末均匀的涂抹在鸡肉上
4. 在400° F下烘烤30分钟

上面是一个做芥末烤鸡的好算法。但是我们需要做的是写一个程序而不是烤鸡，如果我们理解了，上面的伪代码就能转化成下面的实际代码。

```
preheatOven(400);
placeChicken(4,"baking dish"167);
spreadMustard();
bake(400,30);
```

下面是一个利用算法解决数学问题的例子。让我们声明一个算法来计算1到N的总和。

$$\text{SUM}(N) = 1+2+3+\dots+N$$

其中N是任何大于0的整数。

1. 设置SUM=0，计数的时候l = 1
2. 重复下面的步骤同时l小于等于N
 - a. 计算SUM + l，并将结果保存在SUM里面
 - b. l的递增值等于1
3. 答案就是SUM的结果

将上述的算法翻译成代码就是：

```

int sum = 0;
int n = 10;
int i = 0;

while (i <= n) {
    sum = sum + i;
    i++;
}

println(sum);

```

第1步.设置sum = 0, 计数i=0
 第2步.重复while循环 i<=n
 第2a步.递增总和
 第2b步.递增i
 第3步.答案就在sum中, 显示sum

传统上，编程会通过步骤(1)制定一个想法 (2)利用算法实现这一个想法 (3)写出算法的代码。在烤鸡以及求总和的例子中我们已经了解这些步骤。但是有一些想法很复杂很庞大，不可能一步完成。所以我们就需要修改这三个步骤将它变成 (1)制定一个想法 (2)把一个大的想法分解成一些小的部分进行管理 (3)写出各个部分的算法 (4)将算法转化为代码 (5)把所有的算法写出来放在一起 (6)将这些放在一起的代码合为一体。

这并不等于说我们不应该用单一的思想尝试，甚至完全改变原来的想法。而且容易肯定的是，一旦代码完成之后，其他的工作几乎全部都是在整理代码以及修复bug以及添加一些额外的功能。然而正是这种思维过程，能够指导你将想法变成代码。如果你利用这个步骤开发你的项目，创建代码实现你的想法就会没那么困难了。

10.3 从想法到每个部分

为了实践我们的发展步骤，我们将从想法开始，做一个很简单的游戏例子。在这之前，我们需要用文字描述这个游戏。

雨战游戏

这个游戏的目的是在雨滴落到地面之前抓住它。几乎每隔一段实践(取决于游戏的难度等级)，一些雨滴就会从屏幕顶部降落，他们的都是随机的x坐标以及随机的降落速度。玩家的目标就是不要让任何雨点掉落到屏幕底部。

练习10-1：写出你想创建项目的看法，尽量保持简洁，但不要太简单。需要的元素，以及需要做的一些行为(运动)。



现在就让我们看看如何利用“雨战”的想法，并将它分解成多个小部分。我们如何下手呢？首先，我们可以开始思考游戏中的元素：雨点以及捕捉的手。其次，我们应该思考以下这些元素的行为(运动)。例如，我们会需要一个定时机制让雨滴“每隔一段实践降落”，我们同样也需要一个判断机制，来确定雨点是否被“抓住”。让我们更正式的组织这些部分吧。

- 部分1** 开发一个由鼠标控制的圆，这个圆就是玩家控制的“雨滴捕手”。
- 部分2** 开发一个判断机制确定2个圆是否相交。这将用于在玩家是否捕捉到了雨点。
- 部分3** 写一个计时器，让它在每隔N秒执行一次函数。
- 部分 4** 开发一个圆从屏幕顶部向底部降落的程序，这就是降落的雨滴。

从1到3部分都很简单，并且每一个步骤都能一次性完成。但是随着第4部分的开始，就代表了它是一个较大的项目，它复杂得让我们必须将这个运动分解成更小的步骤并进行备份。



练习10-2：将你在练习10-1中写的项目想法分解成几个小部分并且写出来。尽可能让每一部分简单。如果有些部分太复杂，可以进一步分解那写复杂的部分。

章节10.4到10.7将遵循2a,2b和2c的步骤。对于每一个步骤，我们会首先写出算法的伪代码，然后在转换成实际代码，最后完成一个面向对象版本。如果我们正确的做到我们的工作，所有的函数都会放到一个类中，当我们在进行第3步时，它能够很容易的在最后的项目中调用出来。

10.4 Part 1：雨滴捕手

这是最简单的部分，这些都在第三章中学过。写2行伪代码是一个很好的习惯，表明这一部分已经能够直接编写，不需要分解成更小的部分。

伪代码：

- 擦除背景
- 鼠标的位置画一个圆

转化成代码就是：

```
void setup() {
    size(400,400);
    smooth();
}
```

```
void draw() {
    background(255);
    stroke(0);
    fill(175);
    ellipse(mouseX, mouseY, 64, 64);
}
```

这是很好的一步，但是我们并没有做完。如之前所说，我们的目标是开发一个雨滴捕手的面向对象编程。当我们使用这个代码，并把它放入最终的编程中时，我们会想把它单独分离出来归为一类，这样我们就能过做一个“捕手”对象。我们的伪代码就会修改成像下面一样。

Setup:

- 初始化捕手对象

Draw:

- 擦除背景
- 设置捕手的坐标就是鼠标的坐标
- 显示捕手

例10-1的代码显示了一个假设的捕手对象。

例 10-1：捕手

```
Catcher catcher;

void setup() {
    size(400,400);
    catcher = new Catcher(32);
    smooth();
}

void draw() {
    background(255);
    catcher.setLocation(mouseX, mouseY);
    catcher.display();
}
```

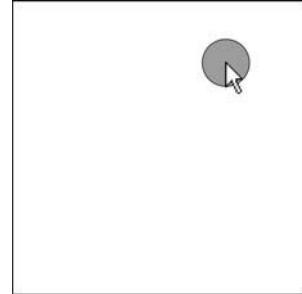


图 10.1

捕手本身的类非常简单，对于坐标以及尺寸都有变量，并且附有2个函数，一个设置了它的坐标，另一个设置了它的显示。

```
class Catcher {
    float r; // radius
    float x,y; // location

    Catcher(float tempR) {
        r = tempR;
        x = 0;
        y = 0;
    }
}
```

```

void setLocation(float tempX, float tempY) {
    x = tempX;
    y = tempY;
}

void display() {
    stroke(0);
    fill(175);
    ellipse(x,y,r*2,r*2);
}
}

```

10.5 Part 2: 相交

在第2部分需要我们确定什么时候捕手和雨滴属于相交。相交的函数就是我们这一步的重点。我们将用一个简单的弹球的类(在例5-6中出现的例子)来确定什么时候属于2隔弹球在相交。在“整合”过程中，这个 intersect() 相交函数会被纳入捕手的类中，去抓捕雨滴。下面时我们对于交点部分的算法。

Setup:

- X和Y的坐标
- 半径
- X及Y方向上的运动速度

Draw:

- 函数构造器
- 如果球#1与球#2相交，这2个球都会变成白色。否则就是灰色

当然，这里的工作只是为了测试相交，我们会在另一个时间使用它。首先，我们需要一个简单的“球”的类。

Data:

- X和Y的坐标
- 半径
- X及Y方向上的运动速度

Functions:

- 函数构造器
 - 基于参数设置半径
 - 选择一个随机的坐标
 - 选择一个随机的速度
- 运动
 - 基于参数设置半径
 - 选择一个随机的坐标
 - 选择一个随机的速度
- 显示
 - 在X和Y坐标上画一个圆

我们现在准备将上面的伪代码翻译成代码。

例 10-2：弹球的类

```
class Ball {
    float r; // radius
    float x,y; // location float
    xspeed,yspeed; // speed

    // Constructor
    Ball(float tempR) {
        r = tempR;
        x = random(width);
        y = random(height);
        xspeed = random(-5,5);
        yspeed = random(-5,5);
    }

    void move() {
        x += xspeed; // Increment x
        y += yspeed; // Increment y

        // Check horizontal edges
        if (x > width || x < 0) {
            xspeed *= -1;
        }

        //Check vertical edges
        if (y > height || y < 0) {
            yspeed *= -1;
        }
    }

    // Draw the ball
    void display() {
        stroke(255);
        fill(100,50);
        ellipse(x,y,r*2,r*2);
    }
}
```

通过这个类，就很容易去创建2个球的对象。最后，在最终的草图中我们会需要一个数组来做出许多雨滴，但是现在，2个球的变量更简单。

例 10-2：2个球的对象

```
// Two ball variables
Ball ball1;
Ball ball2;

void setup() {
    size(400,400);
    smooth();

    // Initialize balls
    ball1 = new Ball(64);
    ball2 = new Ball(32);
}
```

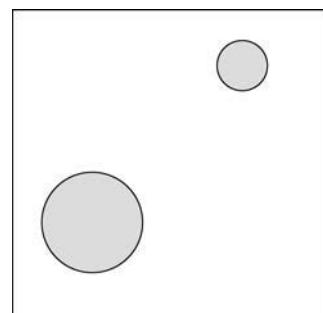


图 10.2

```

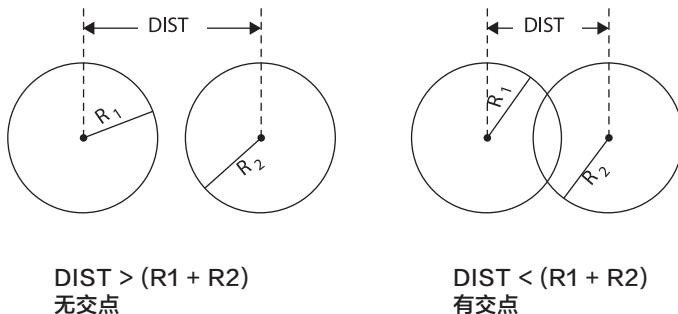
void draw() {
    background(0);

    // Move and display balls
    ball1.move();
    ball2.move();
    ball1.display();
    ball2.display();
}

```

现在我们已经在屏幕上设置好2个球在移动的系统，我们需要一个算法来确定2个圆什么时候属于相交。在Processing中，我们指导我们能够使用dist() 函数计算点与点之间的距离(参考第7章)。我们也需要每一个圆的半径。如10.3中的图表所示，我们可以比较圆之间距离的综合，利用半径长度，确定圆是否有交点。

图 10.3



OK,假设如下：

- $x1, y1$:一个圆的坐标
- $x2, y2$:另一个圆的坐标
- $r1$:一个圆的半径
- $r2$:另一个圆的半径

我们的陈述：

如果 $(x1,y1)$ 和 $(x2,y2)$ 之间的距离小于 $r1$ 和 $r2$ 的和，那么这2个圆就相交。

我们现在的任务就是要基于上面的陈述，写一个返回ture或者false的函数。

```

// A function that returns true or false based on whether two circles intersect
// If distance is less than the sum of radii the circles touch
boolean intersect(float x1, float y1, float x2, float y2, float r1, float r2) {
    float distance = dist(x1,y2,x2,y2); // Calculate distance
    if (distance < r1 + r2) { // Compare distance to r1 + r2
        return true;
    } else {
        return false;
    }
}

```

现在函数已经完整，我们来利用一些数据测试球1和球2.

```
boolean intersecting = intersect(ball1.x,ball1.y,ball2.x,ball2.y,ball1.r,ball2.r);
if (intersecting) {
    println("The circles are intersecting!");
}
```

上面的代码看起来有一些笨拙，但是它却是很有用的一步函数，它整合了球本身的类。让我们先来看看整个主程序。

```
// Two ball variables
Ball ball1;
Ball ball2;

void setup() {
    size(400,400);
    framerate(30);
    smooth();

    // Initialize balls
    ball1 = new Ball(64);
    ball2 = new Ball(32);
}

void draw() {
    background(0);

    // Move and display balls
    ball1.move();
    ball2.move();
    ball1.display();
    ball2.display();
    boolean intersecting = intersect(ball1.x,ball1.y,ball2.x,ball2.y,ball1.r,ball2.r);
    if (intersecting) {
        println("The circles are intersecting!");
    }
}

// A function that returns true or false based on whether two circles intersect
// If distance is less than the sum of radii the circles touch
boolean intersect(float x1, float y1, float x2, float y2, float r1, float r2) {
    float distance = dist(x1,y2,x2,y2);           // Calculate distance
    if (distance < r1 + r2) {                      // Compare distance to r1 + r2
        return true;
    } else {
        return false;
    }
}
```

既然我们在使用面向对象编程的球，那么突然有一个intersect()函数在球的类之外这也不是什么可怕的逻辑。这个球的对象知道怎么样会和另一个球对象相交。我们的代码能够通过整合相交逻辑在类中变得更高级。可以说”ball1.intersect (ball2);”或者 球1相交于球2?

```
void draw() {
    background(0);
    // Move and display balls
    ball1.move();
    ball2.move();
```

```

ball1.display();
ball2.display();

boolean intersecting = ball1.intersect(ball2);
if (intersecting) {
  println("The circles are intersecting!");
}
}

```

假设函数**intersect()**在球的类中，并且有返回值**true**或**false**。

下面这个模型以及算法能够测试相交，这个函数在球的类中。注意函数怎么使用使用它自身的坐标(x和y)以及另一个球的坐标(b.x和b.y)。

```

// A function that returns true or false based on whether two Ball objects intersect
// If distance is less than the sum of radii the circles touch
boolean intersect(Ball b) {
  float distance = dist(x,y,b.x,b.y); // Calculate distance
  if (distance < r + b.r) { // Compare distance to sum of radius
    return true;
  } else {
    return false;
  }
}

```

把它们整合在一起，我们就有了例10-3的代码。

例 10-3：球的相交

```

// Two ball variables
Ball ball1;
Ball ball2;

void setup() {
  size(400,400);
  smooth();
  // Initialize balls
  ball1 = new Ball(64);
  ball2 = new Ball(32);
}

void draw() {
  background(255);
  // Move and display balls
  ball1.move();
  ball2.move();
  if (ball1.intersect(ball2)) {
    ball1.highlight();
    ball2.highlight();
  }
  ball1.display();
  ball2.display();
}

```

```

class Ball {
  float r; // radius
  float x,y;
  float xspeed,yspeed;
  color c = color(100,50);
}

```

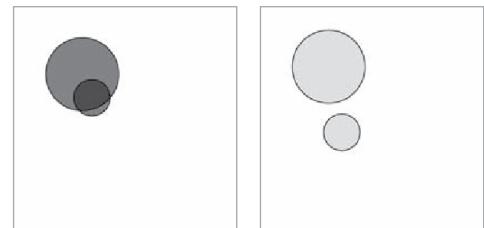


图 10.4

新的！一个对象能够有一个函数将另一个对象作为参数。这时一种对象之间的沟通。在这种情况下它们会判断它们是否相交。

```

// Constructor
Ball(float tempR) {
    r = tempR;
    x = random(width);
    y = random(height);
    xspeed = random(-5,5);
    yspeed = random(-5,5);
}

void move() {
    x += xspeed; // Increment x y += yspeed; // Increment y
    // Check horizontal edges
    if (x > width || x < 0) {
        xspeed *= -1;
    }

    // Check vertical edges
    if (y > height || y < 0) {
        yspeed *= -1;
    }
}

void highlight() {
    c = color(0,150);
}

// Draw the ball
void display() {
    stroke(0);
    fill(c);
    ellipse(x,y,r*2,r*2);
    c = color(100,50);
}

// A function that returns true or false based on whether two circles intersect
// If distance is less than the sum of radii the circles touch
boolean intersect(Ball b) {
    float distance = dist(x,y,b.x,b.y); // Calculate distance
    if (distance < r + b.r) {           // Compare distance to sum of radii
        return true;
    } else {
        return false;
    }
}

```

每当球相交时，highlight()这个函数就会被调用，它们的颜色就会变暗。

在球被显示时，颜色会重置回一个较暗的灰色。

对象可以被传递到函数中作为参数！

10.6 Part 3：计时器

我们下一步的任务就是开发一个每隔N秒执行函数的计数器。同样，我们会分成2个步骤，首先时写出程序的主体，然后根据程序的逻辑将其放进计时器这个类中。Processing拥有函数hour(), second(), minute(), month(), day(), 以及 year()来处理时间。我们可以想像使用second()函数来明确多久一次。但是这不是特别方便，因为second()的翻转时间是每分钟从60到0次。

对于计时器来说，函数millis()是最好的。首先，millis()返回的数字是以毫秒来计算的，它允许更精确更庞大的处理。以毫秒是千分之一秒(1,000 ms = 1 s)。第二，millis()永远不会翻滚回0，因此需要在某一个毫秒数的时刻，减去它之后的时刻，答案永远是其中所经过的时间。

比方说我们希望有一个小程序能够在5秒之后改变背景的颜色为白色。5秒等于5000ms，所以只需要检查millis()的答案是否大于5000。

```
if (millis() > 5000) {
    background(255,0,0);
}
```

如果要让这个程序稍微复杂一点，我们可以让这个程序每隔5秒就变一个随机的背景颜色。

Setup

- 保存开始时间(注意开始时间应该一直为0，但是它保存在变量中是有用的)。我们将它叫做”savedTime”。

Draw

- 计算现在所经过的时间(即millis())减去savedTime。并保存为”passTime”。
- 如果passedTime大于5000，填充一个新的随机背景颜色，并且重置savedTime到现在的时间。这一步会让计时器重新开始计算。

例 10-4：做一个计时器

```
int savedTime;
int totalTime = 5000;

void setup() {
    size(200,200);
    background(0);
    savedTime = millis();
}

void draw() {
    // Calculate how much time has passed
    int passedTime = millis() - savedTime;

    // Has five seconds passed?
    if (passedTime > totalTime) {
        println("5 seconds have passed!");
        background(random(255)); // Color a new background
        savedTime = millis(); // Save the current time to restart the timer!
    }
}
```

有了上面的逻辑，我们能够将计时器放到类中。让我们想想什么数据会涉及到计时器。一个计时器必须知道它什么时候开始的(savedTime)以及它需要运行多久(totalTime)。

Data

- savedTime
- totalTime

Function

- start()
- isFinished()—返回 true 或者false

从上述非面向对象的示例代码以及结构，我们可以总结出例10–5。

例 10–5：面向对象计时器

```
Timer timer;

void setup() {
    size(200,200);
    background(0);
    timer = new Timer(5000);
    timer.start();
}

void draw() {
    if (timer.isFinished()) {
        background(random(255));
        timer.start();
    }
}

class Timer {
    int savedTime; // When Timer started
    int totalTime; // How long Timer should last
    Timer(int tempTotalTime) {
        totalTime = tempTotalTime;
    }
    // Starting the timer
    void start() {
        savedTime = millis();
    }
    boolean isFinished() {
        // Check how much time has passed
        int passedTime = millis()- savedTime;
        if (passedTime > totalTime) {
            return true;
        } else {
            return false;
        }
    }
}
```

当计时器开始时它会存储当前的毫秒时间。

当经过5000ms的时间后，函数
isFinished()就会返回true。计
时器的工作就会重新开始计算。

10.7 Part 4：雨滴

从开始什么都没有，到现在我们创建了一个捕手，我们知道怎么去测试相交，以及开发计时器对象。最后还有一部分就是雨滴。最后，我们想让一组降落的雨滴从屏幕的顶部降落到底部。由于这一步涉及到创建一个移动的对象数组，所以将这一步分解成更小的4个子步骤会更合适。

Part 4 子部分：

Part 4.1 一个单独移动的雨滴

Part 4.2 一组雨滴对象

Part 4.3 灵活的雨滴数量(一个一个的出现)

Part 4.4 漂亮的雨滴外观

部分4.1 创建一个运动的雨滴（对于现在来说就是一个圆）

- 在y坐标上给一个运动的速度
- 显示雨滴

我们已经在下面例10-6中写出了Part 4.1的代码。

例 10-6：一个简单的雨滴降落行为

```
float x,y; // Variables for drop location
```

```
void setup() {
    size(400,400);
    background(0);
    x = width/2;
    y = 0;
}
```

```
void draw() {
    background(255);

    // Display the drop
    fill(50,100,150);
    noStroke();
    ellipse(x,y,16,16);

    // Move the drop
    y++;
}
```

不过，我们需要更深一步并且创建出雨滴的类（毕竟我们希望在最后能够有一组雨滴）。在创建的类中，我们可以多添加几个变量，如速度以及尺寸，以及测试雨滴是否到达屏幕底部的函数，这将在最后游戏计分的时候非常有用。

```
class Drop {
    float x,y; // Variables for location of raindrop
    float speed; // Speed of raindrop
    color c;
    float r; // Radius of raindrop
```

```

Drop() {
    r = 8;                                // All raindrops are the same size
    x = random(width);                    // Start with a random x location
    y = -r*4;                             // Start a little above the window
    speed = random(1,5);                  // Pick a random speed
    c = color(50,100,150);                // Color
}

//Move the raindrop down
void move() {
    y += speed; // Increment by speed
}

// Check if it hits the bottom
boolean reachedBottom() {
    // If we go a little past the bottom
    if (y > height + r*4) {
        return true;
    } else {
        return false;
    }
}

// Display the raindrop
void display() {
    // Display the drop
    fill(50,100,150);
    noStroke();
    ellipse(x,y,r*2,r*2);
}

```

它的降落速度y现在在函数move()中。

它的降落速度y现在在函数move()中。

在我们即将到4.3部分时，我们应该确保一个降落对象是没有任何问题的。作为联手，请完成练习10-3的代码它能测试一个单独降落的对象。

练习10-3：填写下列空白，完成“测试雨滴”的草图

```

Drop drop;
void setup() {
    size(200,200);
    _____;
}

void draw() {
    background(255);
    drop._____;
    _____;
}

```

现在我们这一步要做的已经完成，接下来我们需要的就是将一个降落雨滴编程一组降落雨滴—Part 4.2。这也正是我们在第9章所完善的技术。

```
// An array of drops
Drop[] drops = new Drop[50];
```

设置一个数组，让它拥有50个雨滴

```
void setup() {
    size(400,400);
    smooth();
    // Initialize all drops
    for (int i = 0; i < drops.length; i++) {
        drops[i] = new Drop();
    }
}
```

利用for循环初始化所有的雨滴

```
void draw() {
    background(255);
    // Move and display all drops
    for (int i = 0; i < drops.length; i++) {
        drops[i].move();
        drops[i].display();
    }
}
```

移动以及显示所有的雨滴

但是在上面的代码中有问题，雨滴会一次全部出现在窗口中。根据我们所想的游戏规则，我们想要雨滴一次只显示一次，并且每隔n秒降落一个—现在我们在Part 4.3—灵活的雨滴（一次只显示一次）。我们可以先跳过担心计时器会混乱这些，并且想想让我们在每一帧出现一个新的雨滴降落。我们应该让我们的数组更大一些，允许更多的雨滴。

为了完成这项工作，我们需要一个新的变量来保持跟踪雨滴的总数—“totalDrops”。大多数数组例子为了处理整个列表所以会涉及到整个数组。现在，我们想要访问列表中的一部分，这一部分储存在totalDrops中。让我们一些伪代码来描述以下这个过程：

Setup

- 创建一个保函1000个空间的数组。
- 设置totalDrops = 0。

Draw

- 在数组中创建一个新的雨滴(totalDrops的位置)。由于totalDrops是从0开始的，我们将首先在数组的第一个点创建一个新的雨滴。
- totalDrops在递增(所以下一次我们会到这里来，我们会在下一个数组点创建一个雨滴)
- 如果totalDrops超过了数组的大小，复位到0并且重新开始。
- 移动和显示所有雨滴(即totalDrops)。

例10-7将上面的伪代码转译成了实际代码。

例 10-7：一次显示一个雨滴

```
// An array of drops
Drop[] drops = new Drop[1000];
int totalDrops = 0;

void setup() {
    size(400,400);
    smooth();
    background(0);
}

void draw() {
    background(255);
    // Initialize one drop
    drops[totalDrops] = new Drop();
    // Increment totalDrops
    totalDrops++;
    // If we hit the end of the array
    if (totalDrops >= drops.length) {
        totalDrops = 0; //Start over
    }
    // Move and display drops
    for (int i = 0; i < totalDrops; i++) {
        drops[i].move();
        drops[i].display();
    }
}
```

我们想用新的变量跟踪雨滴的总和

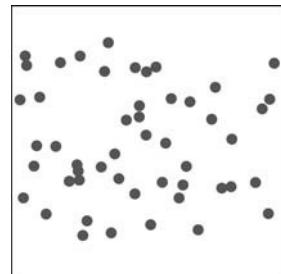


图 10.5

新的！我们不在一次性移动和显示所有的雨滴，而是通过”totalDrops”正确的运行现在的游戏。

我们已经清楚了如何控制时间来表现我们想要的雨滴移动，创建一个类，并且利用这个类创建一个数组。但是从开始以来我们都一直使用圆来显示我们的雨滴。这样做的好处是不用担心我们的绘图代码会延迟并且能够更多的聚焦于运动行为以及组织数据和函数上。而现在我们需要做的就是如何让雨滴的外观更像一个真实的雨滴
Part 4.4—完善雨滴外观。

一种方法是创建一个更像雨滴的圆，开始很小，并且会随着它们向下的运动越来越大。

例 10-8：完善雨滴的外观

```
background(255);
for (int i=2; i<8; i++) {
    noStroke();
    fill(0);
    ellipse(width/2,height/2+i*4,i*2,i*2);
}
```

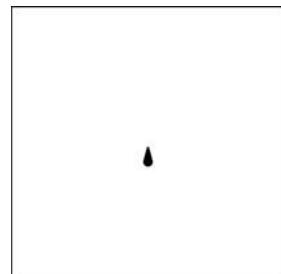


图 10.6

我们可以把该算法整合到雨滴的类中，使用x和y来确定圆开始的位置，并且雨滴的最大半径通过for循环表现。输出如图10.7所示。

```
// Display the raindrop
void display() {
    // Display the drop
    noStroke();
    fill(c);
    for (int i = 2; i < r; i++) {
        ellipse(x,y+i*4,i*2,i*2);
    }
}
```

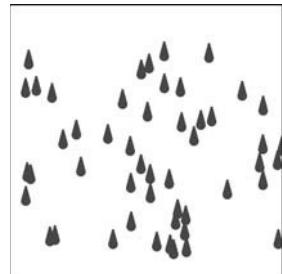


图 10.7

10.8 整合

现在我们已经开发完成了所有的部分，并且确定它们每一个都能完美运行，我们现在需要的就是将它们整合在一个程序中。第一步是创建一个新的Processing草图，并且让它们拥有4个tab，每一个对应每一个的类和主要编程，如图10.8所示。

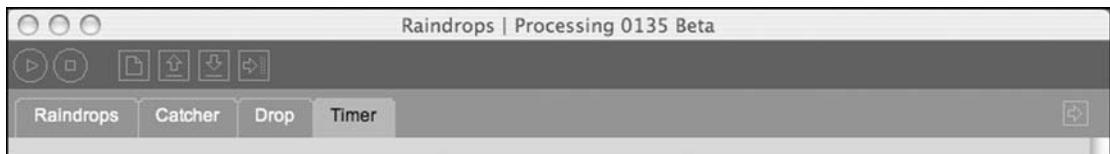


图 10.8

第一步，将所有对应类的代码复制到该tab中。一些个别的不需要变动，所以也每必要让我们重新再写代码。我们需要重新编写的代码是主编程—它们在setup()和draw()中。让我们回到一开始对游戏本身的描述，这样能够让我们更清晰的了解如何整合这些零散的部件，我们可以为这个游戏写一个伪算法。

Setup

- 创建捕手对象
- 创建雨滴数组
- 设置totalDrops等于0
- 创建计时器对象
- 启动计时器

Draw

- 设置捕手的坐标为鼠标的坐标
- 显示捕手
- 移动雨滴
- 显示移动的雨滴
- 如果捕手与雨滴相交，在屏幕中删除雨滴
- 如果计时器完成：增加雨滴的数量并重新开始

请注意，上面每一个单独的步骤都已经在先前的章节制定好了，但是有一个没有：在屏幕中移出雨滴。这个是相当普遍的，在我们计划一个程序的时候也许总会有一些零碎的部分会错过，不过没有关系。这个部分的函数很简单，巧用一些方式，能够让它在我们的整合过程中轻松放进去。

整合的一种方法是将上面所有的算法从一开始就整合在一个草图中，这样就不用担心它们会互相交错。为了这样做，我们需要从每一个部分的全局变量中复制并粘贴它们的setup()和draw()。

下面是一些全局变量：一个捕手对象，一组雨滴对象，一个计时器对象，一个存储雨滴总和的整数。

```
Catcher catcher;           // One catcher object
Timer timer;              // One timer object
Drop[] drops;             // An array of drop objects

int totalDrops = 0;        // totalDrops
```

在setup()中，变量会被初始化。注意，我们可以跳过初始化在数组中单独的雨滴，因为我们已经在同一时间创建了一个。同样我们还需要调用计时器的start()函数。

```
void setup() {
    size(400,400);
    catcher = new Catcher(32);          // Create the catcher with a radius of 32
    drops = new Drop[1000];             // Create 1000 spots in the array
    timer = new Timer(2000);            // Create a timer that goes off every 2 seconds
    timer.start();                     // Starting the timer
}
```

在draw()中，利用对象调用它们的函数。同样我们要按照每一步的顺序粘贴每一部分的代码。

例 10-9：让所有的对象都在一个草图中

```
Catcher catcher;           // One catcher object
Drop[] drops;              // An array of drop objects
Timer timer;               // One timer object
int totalDrops = 0;         // totalDrops

void setup() {
    size(400,400);
    smooth();
    catcher = new Catcher(32);    // Create the catcher with a radius of 32
    drops = new Drop[1000];       // Create 1000 spots in the array
    timer = new Timer(2000);      // Create a timer that goes off every 2 seconds
    timer.start();               // Starting the timer
}
```

```

void draw() {
    background(255);

    catcher.setLocation(mouseX,mouseY);      // Set catcher location
    catcher.display();                      // Display the catcher
}

```

捕手，
来自Part 1

```

// Check the timer
if (timer.isFinished()) {
    println(" 2 seconds have passed! ");
    timer.start();
}

```

计时器，
来自Part 3

```

// Deal with raindrops
// Initialize one drop
drops[totalDrops] = new Drop();
//Increment totalDrops
totalDrops++;
// If we hit the end of the array
if (totalDrops >= drops.length) {
    totalDrops = 0; // Start over
}
// Move and display all drops
for (int i = 0; i < totalDrops; i++) {
    drops[i].move();
    drops[i].display();
}
}

```

雨滴，
来自Part 4

下一步就是将我们已经制定好的这些概念放在一起工作。例如，每隔2秒我们就只会新建一个雨滴(这个由计时器的isFinished() 函数来完成)。

```

// Check the timer
if (timer.isFinished()) {
    // Deal with raindrops
    // Initialize one drop
    drops[totalDrops] = new Drop();
    // Increment totalDrops
    totalDrops++;
    // If we hit the end of the array
    if (totalDrops >= drops.length) {
        totalDrops = 0; // Start over
    }
    timer.start();
}

```

这些对象在一起工作！当计时器“isfinished”，雨滴对象就会被添加一个(通过增加“totalDrops”)。

我们同样也需要确定捕手与雨滴什么时候相交。在章节10.5中，我们通过调用intersect()函数来测试相交。

```
boolean intersecting = ball1.intersect(ball2);
if (intersecting) {
    println("The circles are intersecting!");
}
```

我们可以做很多相同的事，在捕手的类中调用intersect()函数，并且穿过系统中每一个雨滴。我们希望它们在相交时消失。我们可以假设caught()函数能够完成这些内容。

```
// Move and display all drops
for (int i = 0; i < totalDrops; i++) {
    drops[i].move();
    drops[i].display();
    if (catcher.intersect(drops[i])) {
        drops[i].caught();
    }
}
```

对象在一起工作！这里，捕手对象检查它是否与任何雨滴对象相交。

我们的捕手对象原先没有包含函数intersect()，雨滴对象也没有包含caught()。所以这里我们需要写一些新函数作为整合过程中的一部分。

intersect()这个问题很容易解决，因为我们已经在第10.5节中解决，可以直接从捕手的类中复制过来(改变其中的参数，将球对象变成雨滴对象)。

```
// A function that returns true or false based if the catcher intersects a raindrop
boolean intersect(Drop d) {
    // Calculate distance
    float distance = dist(x,y,d.x,d.y);
    // Compare distance to sum of radii
    if (distance < r + d.r) {
        return true;
    } else {
        return false;
    }
}
```

除了调用函数之外，我们还能使用点语法访问一个对象中的变量。

当雨滴被捕获，我们就将它挪到屏幕外的地方(这样它就不会被看见，相当于“消失”)，并通过让速度等于0让其停止移动。虽然我们没有将这个函数用高级的方式整合在一起，但是现在这仍然很简单。

```
// If the drop is caught
void caught() {
    speed = 0; // Stop it from moving by setting speed equal to zero
    y = -1000; // Set the location to somewhere way off-screen
}
```

这样我们就完成了。作为参考，例10-10显示了整个草图。计时器变成了每隔300ms执行一次，让游戏稍微难一些。

例 10-10：雨滴捕手游戏

```

Catcher catcher; // One catcher object
Timer timer; // One timer object
Drop[] drops; // An array of drop objects
int totalDrops = 0; // totalDrops

void setup() {
    size(400,400);
    smooth();
    catcher = new Catcher(32); // Create the catcher with a radius of 32
    drops = new Drop[1000]; // Create 1000 spots in the array
    timer = new Timer(300); // Create a timer that goes off every 2 seconds
    timer.start(); // Starting the timer
}

void draw() {
    background(255);
    catcher.setLocation(mouseX,mouseY); // Set catcher location
    catcher.display(); // Display the catcher
    // Check the timer
    if (timer.isFinished()) {
        // Deal with raindrops
        // Initialize one drop
        drops[totalDrops] = new Drop();
        // Increment totalDrops
        totalDrops++;
        // If we hit the end of the array
        if (totalDrops >= drops.length) {
            totalDrops = 0; // Start over
        }
        timer.start();
    }

    // Move and display all drops
    for (int i = 0; i < totalDrops; i++) {
        drops[i].move();
        drops[i].display();
        if (catcher.intersect(drops[i])) {
            drops[i].caught();
        }
    }
}

class Catcher {
    float r; // radius
    color col; // color
    float x,y; // location
    Catcher(float tempR) {
        r = tempR;
        col = color(50,10,10,150);
        x = 0;
        y = 0;
    }
}

```

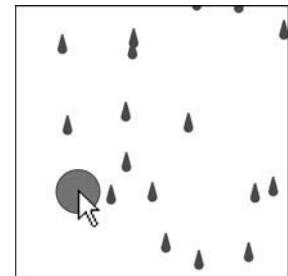


图 10.9

```

void setLocation(float tempX, float tempY) {
    x = tempX;
    y = tempY;
}

void display() {
    stroke(0);
    fill(col);
    ellipse(x,y,r*2,r*2);
}

// A function that returns true or false based if the catcher intersects a raindrop
boolean intersect(Drop d) {
    float distance = dist(x,y,d.x,d.y); // Calculate distance
    if (distance < r + d.r) { // Compare distance to sum of radii
        return true;
    } else {
        return false;
    }
}

class Drop {
    float x,y;           // Variables for location of raindrop
    float speed;         // Speed of raindrop
    color c;             // Color of raindrop
    float r;              // Radius of raindrop

    Drop() {
        r = 8;           // All raindrops are the same size
        x = random(width); // Start with a random x location
        y = -r*4;          // Start a little above the window
        speed = random(1,5); // Pick a random speed
        c = color(50,100,150); // Color
    }

    // Move the raindrop down
    void move() {
        y += speed; // Increment by speed
    }

    // Check if it hits the bottom
    boolean reachedBottom() {
        if (y > height + r*4) { // If we go a little beyond the bottom
            return true;
        } else {
            return false;
        }
    }

    // Display the raindrop
    void display() {
        // Display the drop
        fill(c);
    }
}

```

```

noStroke();
for (int i = 2; i < r; i++) {
    ellipse(x,y + i*4,i*2,i*2);
}
}

// If the drop is caught
void caught() {
    speed = 0; // Stop it from moving by setting speed equal to zero
    // Set the location to somewhere way off-screen
    y = -1000;
}

class Timer {
    int savedTime; // When Timer started
    int totalTime; // How long Timer should last
    Timer(int tempTotalTime) {
        totalTime = tempTotalTime;
    }

    // Starting the timer
    void start() {
        savedTime = millis();
    }

    boolean isFinished() {
        // Check out much time has passed
        int passedTime = millis() - savedTime;
        if (passedTime > totalTime) {
            return true;
        } else {
            return false;
        }
    }
}

```



练习10-4：让游戏实施积分制比赛，玩家一开始有10分。每一次雨滴降落到底部就会减少1分。如果所有1000个雨滴降落后还没有到达0分，那么就会开始一个新的等级并且出现的比之前更快。如果在任何一个等级有10个雨滴降落到底部，那么玩家就输了。分数会在屏幕中的一个矩形上显示。
不要试图一次性完成这些内容。一步一步的来做。

10.9 准备行动2

本章的重点不是去学习怎么编写捕手游戏的编程，而是学习一种开发编程的方法—首先说出想法，然后把一个抽象的想法分解成几个部分，并写出这几个步骤的伪代码然后一个一个解决。

最终要的是请记住完成这个流程是需要经过时间的积累以及不断的实践的。每一个初学编程的人都会经历这个过程。

在我们开始这本书其余部分之前，让我们来花一点实践来思考以下我们学到了什么。在这10章中，我们将重点完全集中于编程的基本知识。

- Data—变量和数组
- Control Flow—条件语句及循环语句
- Organization—函数及对象

这些理念不仅仅能运用在Processing中，它还能运用在其他所有编程语言和环境中，如C语言,Actionscript(Flash语言)以及服务器端编程语言PHP。虽然语法可能会有些差别，但是基本理念是不会变化的。

让我们在学习更高级的内容之前，我们将学习如何快速浏览并处理你代码中的错误(第11章：调试)以及如何使用Processing的库(第12章)。很多高级的内容需要通过倒入Processing或者第三方库来完成。Processing的强大优势之一就是能够很容易的添加库进行扩展。在这本书的最后一节也会告诉你如何去创建你自己的库。



第五课项目

步骤1 利用简单的形状绘制以及基本的编程知识开发一套项目。如果你觉得没头绪，试一试乒乓球的游戏。

步骤2 按照本章中所学到的步骤，将这个大的想法分解成小的部分，并一个一个击破。请务必在每个部分中使用面向对象编程。

步骤3 把所有分解的部分最后整合成一个程序。不要忘记任何元素或者函数。

使用下面的提供的空间为你的项目写一些草图。

11. 调试

“他合适的单词和几乎是正确的字之间的差异的区别是闪电和雷电错误之间。”

— Mark Twain

bug发生

5分钟前你的代码完美完成，并且你发誓你所作的是改变一些对象的颜色！但是现在，当飞船撞击小行星，它没有发生任何旋转。但是就在5分钟之前，它还在旋转！你的朋友和说：我看到在转的飞船了，它非常酷！rotate()函数在那。发生了什么事？它应该继续工作。者肯定的电脑的鼓故障。

无论你花多少时间去学习计算机科学，阅读编程书籍，或者不断的泡在编程中，但是总会没办法避免出现编程上的bug。

这真的让人很闹心。

bug可以是程序中的任何缺陷。有时候在Processing中会有很明显的提示(或者根本无法运行)，并在消息窗口显示错误提醒。这种类型的bug能够显示是因为它们都是一些简单的基本错误，变量没有初始化，或者字母错误，或者寻找数组中一个不存在的变量等等。对于一些bug的更多信息，可以看看这本书结尾部分附录中的bug信息。

bug同样也有很隐藏的那种。如果你的Processing草图中的函数运行有问题，这里可能就出现了一个bug。在这种情况下，你的草图可能不会显示任何错误消息。要寻找到这种类型的bug比较困难，因为它不会像之前的代码那样明显。

在本章中，我们会讨论一些关于修复bug(调试)的基本对策。

11.1 提示1：休息一下

严肃点。从电脑跟前走开，去睡觉，慢跑，或者吃一个桔子。做一些编写代码之外的事。我不得不说已经不知道多少次我几个小时无法修复的代码，第二天早上醒过来5分钟内就能解决。

11.2 提示2：让另一个人帮你

把这个问题和朋友说。把你的Processing代码给另一个程序员看(甚至非程序员也可以)，并且通过逻辑来梳理一次，往往能够很好的揭露这些bug。在许多情况下，它是很明显的但是你没有看见，因为你会自我感觉你的代码没有问题。在给别人解释你的代码时，这个检查过程会变慢，所以会很清晰的查看到你的bug。如果你身边没有任何人，你也可以对自己大声说。是的，这样做也许很傻，但是它能够帮助你。

11.3 提示3：简化

简化！简化！简化！

在第10章中，我们将重点放在了渐进式的开发过程中。一步一步开发你的项目，这样易于管理，错误就会更少。当然，有时候也没有办法避免一些bug，所以当它们出现时，渐进式的理念同样也有助于调试。将代码分成小的部分，并且一步一步的调试bug。

做到这一点的方法之一是注释掉大块的代码用于隔离特定部分。下面是一个主要标签草图例子。草图中含有一个蛇的对象，一个按钮对象以及一个苹果对象。(类的代码没有包含在内)。让我们假设所有草图都是正常工作，但苹果是看不见的。要调试这个问题，就需要注释掉一些代码，并且只显示苹果对象。

```
// Snake[] snakes = new Snake[100];
// Button button;
Apple apple;

void setup() {
    size(200,200);
    apple = new Apple();
    /*for (int i = 0; i < snakes.length; i ++ ) {
        snakes[i] = new Snake();
    }
    button = new Button(10,10,100,50);
    smooth();*/
}

void draw() {
    background(0);
    apple.display();
    // apple.move();

    /*for (int i = 0; i < snakes.length; i + + ) {
        snakes[i].display();
        snakes[i].slither();
        snakes[i].eat(apple);
    }
    if (button.pressed()) {
        applet.restart();
    }*/
}

/*void mousePressed() {
    button.click(mouseX,mouseY);
}*/
```

只留下苹果对象的代码，并且只显示苹果对象。通过这种方式我们可以肯定不是其他代码的bug。

大的代码块可以通过/* 和 */注释掉。它们之间的内容全部都是注释掉的内容。

一旦所有的代码被竹石雕，结果可能会有2种。如果苹果没有显示，那么问题一定出现在苹果对象本身，那么接下来就是要在display()函数种找到bug，并且修复它。

如果苹果显示了，那么这个bug可能就是其他代码所引起的。也许是move()函数将苹果放到了屏幕外面，所以我们看不到。又肯呢干是蛇对象意外的覆盖了苹果对象。要弄清楚这一点，我建议你一行一行的减少注释代码。每次减少一行，并且运行草图，并且查看苹果在那一行之后消失。只要消失了，说明你就找到了罪魁祸首，并且可以修复它了。拥有一个面向对象编程能够很好的帮助你调试。另一种尝试是你可以新建一个草图，并且只使用一个类，测试基本的函数。换句话说，不用管你的整个程序。首先创建一个新的关于类的草图。比方说蛇出现错误，为了简化可发现bug，我们可以创建一个草图，只用一个蛇，没有其他的任何对象，这样检查bug会更容易。

```
Snake snake;

void setup() {
    size(200,200);
    snake = new Snake();
}

void draw() {
    background(0);
    snakes.display();
    snakes.slither();
    //snakes.eat(apple);
}
```

由于这个版本不包含苹果对象，我们不能使用这行代码。因为是调试种的一部分，所以我们可能要在这一行增加注释，让它跳过。

虽然我们现在的例子还没有开始涉及到外部设备(我们将在后面的章节中学到)，但是在调试的时候，同时也应该记住到时候要关闭连接这些设备，摄像头，麦克风，用”虚拟”的信息来代替。例如，利用jpg来代替一个视频，或者利用一个本地文件代替网络上的文件。如果问题消失，你就可以马上可以断定你的服务器可能坏了，或者你的相机坏了。如果问题还在，你就可以调试其他代码。如果你担心你的代码问题会恶化，你可以在删除某些函数之前备份一次你的代码。

11.4 提示4：println() 一样是你的朋友

使用消息窗口显示变量值是一个非常有帮助的解决方式。如果一个对象完全在屏幕上失踪，你像知道为什么，你就可以print出它的位置变量的值。它可能看起来就像这样：

```
println("x: " + thing.x + " y: " + thing.y);
```

```
x: 9000000  y: -900000
x: 9000116  y: -901843
x: 9000184  y: -902235
x: 9000299  y: -903720
x: 9000682  y: -904903
```

由上开来，这些坐标值都是一些不合理的坐标值。所以在计算对象坐标的时候可能某个位置出现了问题。如果这个值合理，那么就继续向下检查。或者是什么其他的原因。

```
println("brightness: " + brightness(thing.col) + " alpha: " + alpha(thing.col));
```

结果就是：

```
brightness: 150.0 alpha: 0.0
```

如果对象的透明度值为0，那么就可以很好的解释你为什么看不到这个对象了。我们应该花点时间记住提示3：简化。如果我们先在Processing中简化草图再print这些变量值，这样会简单很多。这样我们就可以肯定是不是这一个类出现了问题，将东西意外的画在画面外。

你也可能注意到上面print出来的陈述会串联出一段文字(详见第17章串联)。作出这一点通常是非常好的习惯。下面的这行代码只有print的值x，没有人和解释。

```
println(x);
```

这样会让你在看这些消息的时候很困扰，特别是你要print多个不同的值的时候，很难分辨什么是什么。如果能把自己的注释放在println()中，这样能够减少很多困扰：

```
println("The x value of the thing I'm looking for is: " + x);
```

此外，println()能够被用于指示目标部分的代码是否已经到达。例如在我们的”弹跳球”例子中，球在窗口右侧的时候不能够正常的反弹，这个问题可能是a你没有明确它什么时候会碰到窗口边缘或者在碰到窗口边缘的时候球在做错误的动作。如果想知道你的代码碰到窗口边缘是否正确的运行，你可以写：

```
if (x > width) {
    println("X is greater than width. This code is happening now!");
    xspeed *= -1;
}
```

如果你在运行草图中没有print出任何消息，那么你的布尔表达式可能是存在问题的。

诚然，println()不是一个完美的调试工具。它很难在信息窗口中跟踪多个信息。它同样也会减慢你运行的流畅度。更高级的开发环境经常会提供调试工具，它允许你跟作特定的变量，暂停程序，提前一行代码等等。这也是我们使用Processing的一个取舍。它很方便使用，但是它不具备一些高级的功能。尽管如此，但是在调试方面，稍有常识的人，也还是能轻松应对这些bug的。

12. 库

“如果真即是美，怎么来的，没有人有他们的头发在图书馆做吗。”

— Lily Tomlin

接下来的很多章节都会需要用到Processing的库。这一章将保函下载，安装以及使用库的方法。我建议你为了能使用基本的库(在第14章中我们会谈论：平易和旋转)来好好阅读本章。当你成功下载到一个库时请回到这里(首次发生例子在第15章：Video)。

12.1 库

每当我们调用Processing函数，如line(), background(), stroke()，等等，我们都是从我们从Processing reference 中找到的(也许也含有这本书的知识)。reference列举了所有在Processing库中可能会用到的函数。在计算机科学中库指的是一些“帮助”代码。一个库可能保函函数，变量以及对象大部分的东西都是来自于Processing的库。

在大多数编程语言中，你需要在代码顶部注明你打算使用的库。这样会告诉编译器(详见第2章)在哪里能够找到这些东西，将你的变成语言翻译成机器代码。如果你自己查看Processing应用程序目录立的文件，你会发现一个在lib文件夹下的名为core.jar的代码。这个jar文件包含了所有我们能在Processing中已经编译后的代码。因为它会用于到每一个项目，所以Processing会假设它一开始就是导入状态，也不需要你明确写出import语句。但是，如果没有这样默认设置的话，你可能需要在每一个草图前都要在顶部加入下列代码：

```
import processing.core.*;
```

“Import” 表明我们打算使用一个库，库的导入会看到“processing.core”，“.” 是通配符，表明我们需要获得库中的一切。库的命名会使用点语法，这样会形成一个JAVA变成语言的类别。当你想获得更多的内容时，你可能要导入其他的库。但是现在，我们只需要知道“processing.core” 是核心库的名称即可。

虽然核心库包含了所有的基础知识，但是对于其他高级的功能，你必须导入指定的库去完成。在第14章中我们会有导入一个特定的库，为了使用OpenGL渲染，我们需要导入OpenGL库：

```
import processing.opengl.*;
```

许多后续章节将需要显式使用外部处理库，如视频，网络，接口，等等。这些库文件，可以发现在<http://www.processing.org/reference/libraries/>在处理Web站点。在那里，你会发现一个列表来处理的图书馆，以及在网络上可供下载的第三方库的链接。

12.2 内置库

使用内置库很简单，因为它不需要安装。这些库(下面是可能用到的一些列表)不是非常多，但是能够由少数在这本书中会提到。

- **Video**— 用于捕捉相机图片，播放影视文件以及纪录电影文件。在第16章以及第21章会提到。
- **Serial**— 用于Processing与外部设备接口之间的传播。在第19章中会提到。
- **OpenGL**— 用于加速草图的图形渲染。在第14章中会提到。
- **Network**— 用于通过网络创造客户端以及服务器之间的传播。在第19章中会提到。
- **PDF**— 用于创建在Processing图形处理过程中所产生的高分辨率PDF文件。在第21章中会提到。
- **XML**— 用于导入来自XML文件的数据。在第18章中会提到。

这些例子是专门针对上面的库所展开的，你能够在目录中找到。Processing网站也有关于这些库的好的文件(在“libraries”页面中能够找到)。他们之间唯一的通用部分就是每次当你使用这些内置库的时候你都需要写出import语句并且放在编程顶部。如果你选择SKETCH→IMPORT LIBRARY 这些语句会自动被添加到草图中。或者你可以简单地手动输入代码(使用导入库菜单选项时不不会做任何其他事，只仅仅会添加import语句)。

```
import processing.video.*;
import processing.serial.*;
import processing.opengl.*;
import processing.net.*;
import processing.pdf.*;
import processing.xml.*;
```

12.3 第三方库

第三方库(也被称为“贡献库”)就好像旷野的西部。在这本树种含有47个第三方库，从声音声称以及分析的能力到数据包嗅探，再到物理模拟，图形界面控制。在书的余下部分也会涉及到这几个三地方库的学习。在这里我们看看下载以及安装第三方库的过程。

你需要做的第一件事就是要找出你安装Processing的位置。在Mac,应用最有可能在“Applications”目录中；在PC,可能在“Program Files”中，这只是一个假设，当然你也可能安装在别处(Processing可以安装在任何目录中，只要能运行就好)，只需要将路径列表变成我们的文件路径即可。

一旦你明确了Processing的安装位置，打开Processing文件夹看一看，如图12.1所示。

/Applications/Processing 0135/

或

c:/Program Files/Processing 0135/

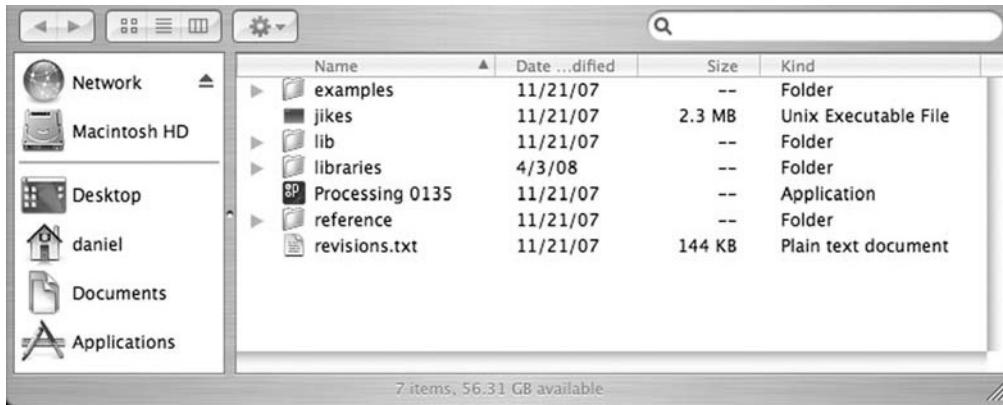


图 12.1

在库的目录下你会发现每个内置库的文件夹以及一个” howto.txt” 文件(提供相关的提示和关于库的说明)，如图12.2所示。

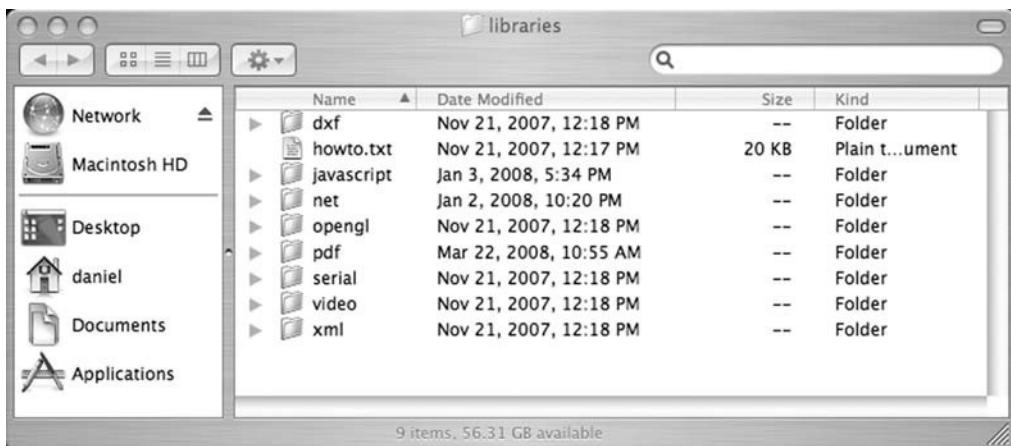


图 12.2

在库的目录中你能安装第三方库。

你会在第18章中看到第一次使用第三方库的例子。” simpleML ”库，用于HTML和XML数据检索简单化，可以在本书的网站上下载到<http://www.learningprocessing.com/libraries>。如果你想直接跳到18章，下载simpleML.zip并且按照下面说明进行安装，如图12.3所示。安装其他第三方库的过程也是一样的，除了名称不同。

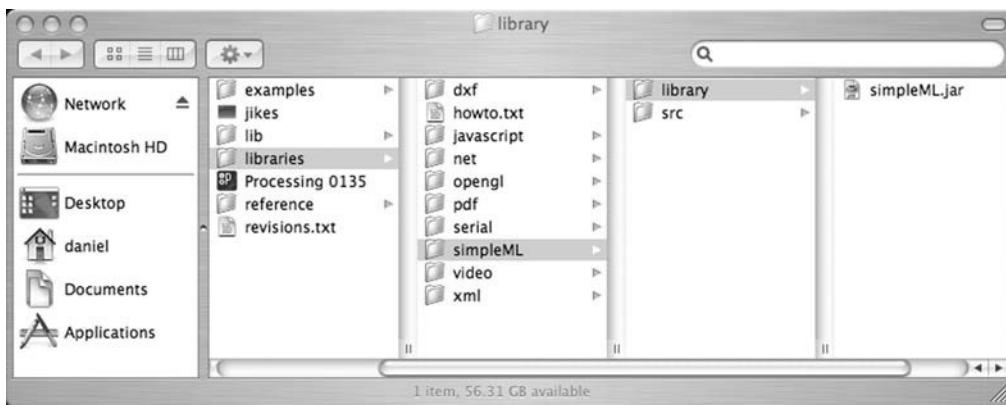


图 12.3

步骤1 解压ZIP文件，这个通常可以通过双击文件或者利用任何解压缩软件打开，如PC机上的Winzip。

步骤2 复制解压后的文件到Processing库的文件夹中。大多数你下载的库会自动解压一个正确的目录结构。完整的目录看起来就像这样：

```
/Processing 0135/libraries/simpleML/library/simpleML.jar
```

通常的都是这样：

```
/Processing 0135/libraries/libraryName/library/libraryName.jar
```

有一些库可能包含一些额外的文件，以及源代码(通常会存储在一个目录中，在“libraryName”文件夹下)你可以手动创建这些文件夹(使用新建或者资源管理器)，并且将libraryName.jar文件放置在相应的位置。

步骤3 重启Processing。如果在第二步的时候Processing还在运行，你需要关闭Processing，并且重新启动才能确定库被识别。一旦你已经重启，如果一切按计划进行，库会出现在“Sketch→Import Library”选项中，如图12.4所示。

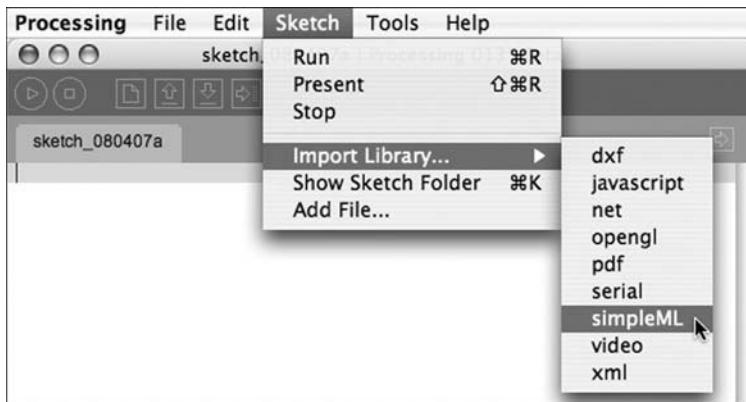


图 12.4

现在新安装的库已经出现在列表中！一旦安装好库，需要做什么就取决于你自己了。在第18章中的例子会在代码中使用第三方库(simple, Yahoo API)以及第20章(Sonia,Minim)。

第六课

世界围绕着你

13.数学

14.平移和旋转（在3D!）

12. 数学

“如果人们不相信数学是简单的，那么他们就完全没意识到生命事是多么复杂。”

— John von Neumann

本章内容：

- 概率
- Perlin 噪音
- 三角函数
- 递推

我们在这里。基本的理论知识已经完成，现在我们即将开始寻找一些更加复杂的东西。从一章到另一章，你可能会发现有一个故事的逻辑。虽然概念并不一定能够流畅的交织在一起，因为他们和之前的一步一步学习不同，但是都很有用。

我们所作的一切就从这里开始，仍然会采用相同的结构流程setup() 和 draw()。我们将继续使用Processing库中的函数以及条件语句和循环的算法，还有利用面向对象变成整合草图。在这一点上，如果有不懂的，我希望你能回到之前的章节回顾一下，如果有需要。

13.1 数学和编程

每当老师叫你在黑板上写出数学代数的答案时，你会不会觉得头上的汗滴开始不断冒出？谈”微积分”色变会不会导致你四肢颤抖？

放松一些，没有必要畏惧，这没有什么可害怕的，但是数学本身却令人害怕。可能你在开始阅读本书的时候你会害怕计算机编程，你会害怕接触代码相关的那种感觉现在是不是已经被宁静所取代？本章旨在采取一个轻松，友好的态度，从数学中使用一些有用的方式，帮助我们开发Processing项目。

你知道，我们其实一直都在使用数学。

例如，自从我们学习了变量，我们可能每一页都有一个代数表达式。

```
x = x + 1;
```

在最近的学习中，第10章，我们通过使用勾股定理判断2个对象之间的交点。

```
float d = dist(x1,x2,y1,y2);
```

这些只是一些例子，随着我们的学习我们会看到越来越多的高级例子，你可能会发现你会在深夜在线，google搜索”反正弦螺旋曲线”。现在我们即将开始有选择性的使用数学。

13.2 取模

我们开始关于取模运算符的讨论，在Processing中它被写成一个百分号。取模是一个很简单概念(当你第一次学习除法的时候你没有学习到它的名称)，它会保留一个数字的余数(一个在屏幕上的形状，一个带有数组的索引值)。系数运算符计算出一个数减去另一个数的余数。他的运行会需要整数以及浮点数。

这里有一些空白需要你来填写完整

20除以6等于3余2 ($6*3+2=18+2=20.$)

因此：

20系数6等于2 ($20\%6=2$)

17除以4等于4余1	$17\%4=1$
3除以5等于0余3	$3\%5=3$
10除以3.75等于2余2.5	$10.0 \% 3.75=2.5$
100除以50等于_____余数_____	$100\%40=_____$
9.25除以0.5等于_____余数_____	$9.25 \% 0.5=_____$

你会发现，如果 $A=B\%C$ ，A永远不会大于C.余数永远不能大于除数。

0%3=0
1%3=1
2%3=2
3%3=0
4%3=1
等等

因此当你需要一个循环计数器让变量到0的时候，你可以使用取模。如下面的代码所示：

```
x = x + 1;
if (x >= limit) {
x = 0; }
```

我们可以替换成：

```
x = (x + 1) % limit;
```

如果你想在一个时间使用数组中的一个元素这是非常有用的，当你得到数组的长度时，它总是可以返回到0。

例 13-1：取模

```
// 4 random numbers
float[] randoms = new float[4];
int index = 0; // Which number are we using

void setup() {
    size(200,200);
    // Fill array with random values
    for (int i = 0; i < randoms.length; i++) {
        randoms[i] = random(0,256);
    }
    frameRate(1);
}

void draw() {
    // Every frame we access one element of the array
    background(randoms[index]);
    // And then go on to the next one
    index = (index + 1) % randoms.length;
}
```

使用取模运算周期计数器回到0。

13.3 随机数字

在第4章中，我们介绍了random()函数，它能够使我们能够随机填写一个变量值。Processing中的随机数通常在某个区间之内随机产生。例如，我们想要一个0到9之间的随机整数，0有10%的几率被随机到，1也有10%的几率被随机到，2也有10%的几率被随机到，以此类推。我们可以写一个简单的数组草图证明这个事实。如例13-2所示。

伪随机数

我们从random()函数中所得到的随机数不是真正的随机，并且它被称为“伪随机”。它是利用了数学函数模拟随机性所得出来的结果。它会随着时间的推移形成一个规律。但是这个时间会需要很久，所以说它对于我们来说和纯随机还是基本相同的。

例 13-2：随机数分配

```
// An array to keep track of how often random numbers are picked.
float[] randomCounts;

void setup() {
    size(200,200);
    randomCounts = new float[20];
}

void draw() {
    background(255);
    // Pick a random number and increase the count
    int index = int(random(randomCounts.length));
    randomCounts[index] ++;

    // Draw a rectangle to graph results
    stroke(0);
    fill(175);
    for (int x = 0; x < randomCounts.length; x++) {
        rect(x*10,0,9,randomCounts[x]);
    }
}
```

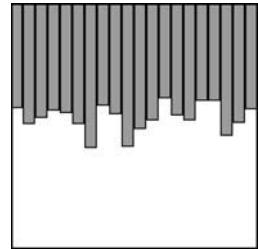


图 13.1

这里有一些小技巧，我们可以使用random()来产生一个均匀分布的随机数让某些事情发生。例如我们想创造一个有10%几率是绿色和90%几率是蓝色的背景草图会怎么做？

13.4 概率的回顾

让我们来回顾一下概率的基本原则，首先看到的是单一事件概率，也就是事情发生的可能性。

给予一个特定的数量算出可能的结果，任何可能发生的概率是需要发生这个事的数量除以该事件的总数。最简单的例子就是抛硬币。总共有两种结果(正面或反面)。有人头的一面是需要计算的概率，因此头的概率等于那1面除以2面总数，也就是50%。

再想想一套52张的牌。A(有4张)在牌中出现的几率是：

王牌的数量除以牌的总数=4/52=0.077=~8%

有红方牌的几率是：

红方牌的数量除以牌的总数=13/52=0.25=25%

我们也可以计算出在多个事件中几个事件作为一个整体发生的概率。

抛硬币三次，出现一次头面的概率是：

$$(1/2) * (1/2) * (1/2) \text{ 1/8 (or 0.125).}$$

换句话说，一个硬币抛3次出现的概率是八分之一。



13.5 事件概率代码

这里有一些用于random()函数和概率代码的使用技巧。例如，如果我么你选择一些数字填充数组（有一些重复的），我们可以从中随机挑选并且通常事件都基于我们所选择的。

```
int[] stuff = new int[5];
stuff[0] = 1;
stuff[1] = 1;
stuff[2] = 2;
stuff[3] = 3;
stuff[4] = 3;

int index = int(random(stuff.length));
if (stuff[index] == 1) {
    // do something
}
```

在一个数组中选择一个随机元素

如果你运行这个代码，有40%几率会选择的值1，20%的几率会选择的值为2,还有40%的几率会选择的值为3.

另一种策略就是要一个随机数(简单起见，我们只考虑0到1之间的浮点数)并且只如果我们在一定范围内挑选随机数它一定会发生事件。

```
float prob = 0.10; // A probability of 10%
float r = random(); // A random floating point value between 0 and 1

if (r < prob) { // If our random is less than .1
    /*INSTIGATE THE EVENT HERE*/
}
```

这些代码有10%的几率会被执行

同样的方法也可以应用余多种结果

Outcome A — 60% | Outcome B — 10% | Outcome C — 30%

为了实现这个代码，我们可以挑选一个随机浮点数，来检查它到底是落在哪里。

- Between 0.00 and 0.60 (10%) → outcome A.
- Between 0.60 and 0.70 (60%) → outcome B.
- Between 0.70 and 1.00 (30%) → outcome C.

例13-3中的圆有3中不同的颜色，每个颜色都有不同的几率出现(red: 60%, green: 10%, blue: 30%)。这个例子显示如图13.2所示。

例 13-3：概率

```

void setup() {
    size(200,200);
    background(255);
    smooth();
    noStroke();
}

void draw() {
    // Probabilities for 3 different cases
    // These need to add up to 100%!
    float red_prob = 0.60; // 60% chance of red color
    float green_prob = 0.10; // 10% chance of green color
    float blue_prob = 0.30; // 30% chance of blue color
    float num = random(1); // pick a random number between 0 and 1

    // If random number is less than .6
    if (num < red_prob) {
        fill(255,53,2,150);
    // If random number is between .6 and .7
    } else if (num < green_prob + red_prob) {
        fill(156,255,28,150);
    // All other cases (i.e. between .7 and 1.0)
    } else {
        fill(10,52,178,150);
    }
    ellipse(random(width),random(height),64,64);
}

```

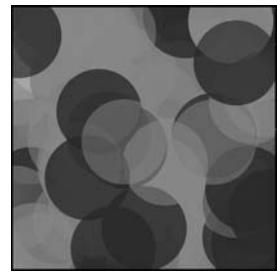


图 13.2



练习13-2：在一组牌中出现2个A的几率是多少？

```
float y = 100;
```

```
void setup() {
    size(200,200);
    smooth();
}
```

```
void draw() {
    background(0);
    float r = random(1);
```

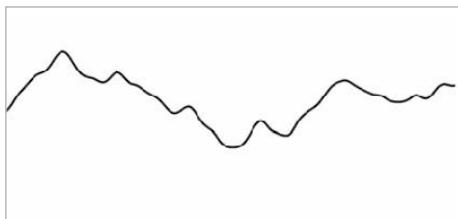
```
    ellipse(width/2,y,16,16);
}
```

13.6 Perlin 噪音

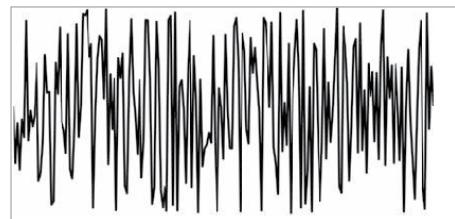
一个好的随机数产生就是看起来与数字没有任何关系。如果它们没有表现出明显的规律，它们会被认为是随机的。

在编程的行为中拥有一个有机的以及高质量的，又有一点点随机的事是好的。但是我们不希望随机性太大，这是由Ken Perlin在80年代初所开发的一个叫”Perlin noise”的函数，它能够自然排序声称一个伪序列随机数。它最初被设计用来创建程序纹理，Ken Perlin 也同时获得了奥斯卡技术成就奖。Perlin 噪音可以用来产生各种有趣的特效，包括云，风景，大理石纹理等等。

图 13.3



Perlin 噪音



普通随机

噪声详细

如果你访问网站Processing.org 噪声reference,你会发现噪音能够计算出8个音节来。你可以改变音节的数字以及利用noiseDetail()函数来进行相关的应用。详细请参考http://processing.org/reference/noiseDetail_.html

你也可以阅读更多关于Ken Perlin自己说的有关噪音的工作原理：<http://www.noisemachine.com/talk1/>

Processing拥有一个内置的关于Perlin噪音的算法函数noise()。函数noise()会采用一个两个或三个参数(参考noise计算出的” space”，一个两个或三个维度)。这一章我们只会接触到一维。你可以访问Processing网站详细了解2维以及3维。

1维Perlin噪音会随着时间推移产生一个线性的顺序。例如：

0.364, 0.363, 0.363, 0.364, 0.365

注意数字是怎么随机的加减的，它们会很靠近前一个的值。现在，为了得到这些数字的处理方式，我们必须做2件事：(1) 调用函数noise()，(2) 将当前的次数作为参数，我们通常会让次数为 t=0 开始，因此调用函数就像这样:” noise(t);”

```
float t = 0.0;
float noisevalue = noise(t); // Noise at time 0
```

我们也可以采用上面的代码并且让它在draw()中循环。

```
float t = 0.0;
void draw() {
    float noisevalue = noise(t);
    println(noisevalue);
}
```

0.28515625
0.28515625
0.28515625
0.28515625

上面代码的结果都会重复相同的值。这是因为我们在noise()函数中重复相同的” 次数”，0.0，一次又一次。如果增加” 次数” 变量t，结果就会不同。

```
float t = 0.0;
void draw() {
    float noisevalue = noise(t);
    println(noisevalue);
    t += 0.01;
}
```

0.12609221
0.12697512
0.12972163
0.13423012
0.1403218

次数在向前递增

怎样快速地让我们递增同样也影响噪声的平滑度。可以试一试将递增的值改为0.01, 0.02, 0.05, 0.1, 0.0001等等。

到现在位置，你可能已经注意到noise()始终会返回一个0到1之间的浮点数。这个细节是不能忽视的，它会影响到我们如果在Processing中使用到Perlin噪音。例13-4分配了一个noise()函数的结果给圆的尺寸。noise值的增减乘以窗口的宽度。如果窗口的宽度为200, noise()的范围在0.0到1.0之间，那么noise()乘以窗口宽度的结果就是0.0到200.。下面列表以及例13-4说明了这点。

Noise 值	乘数	等于
0	200	0
0.12	200	24
0.57	200	114
0.89	200	178
1	200	200

例 13-4：Perlin 噪音

```
float time = 0.0;
float increment = 0.01;

void setup() {
    size(200,200);
    smooth();
}

void draw() {
    background(255);
    float n = noise(time)*width;
    // With each cycle, increment the "time"
    time += increment;
    // Draw the ellipse with size determined by Perlin noise
    fill(0);
    ellipse(width/2,height/2,n,n);
}
```

noise 的值能够随着窗口的宽度随机缩放。

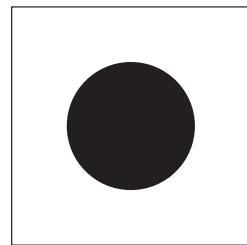


图 13.4

练习 13-3：利用 Perlin 噪音设置圆的位置，并运行代码是否显示圆会很“自然”的移动？

```
// Noise "time" variables
float xtime = 0.0;
float ytime = 100.0;
float increment = 0.01;
void setup() {
  size(200,200);
  smooth();
}
void draw() {
  background(0);
  float x = _____;
  float y = _____;
  _____;
}

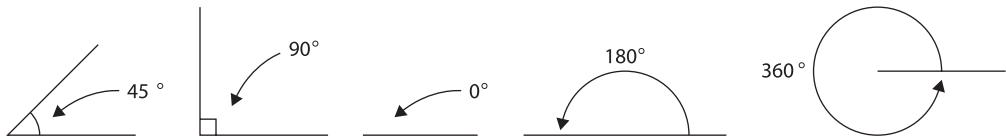
// Draw the ellipse with size determined by Perlin noise
fill(200);
ellipse(_____,_____,_____,_____);
}
```

13.7 角度

在这本书中的一些例子我们需要明确知道角度的定义以及基本的运用。在第14章中，例如我们需要知道知道怎么使用rotate()函数来调节旋转和旋转的对象以适合良好的体验。

为了迎接即将来到的例子，我们需要学习了解弧度和度。这可能是你熟悉的一个角度的概念。一个完整的旋转是从0到 360° 。一个度是 90° （直角），如图13.5中第二个所示。

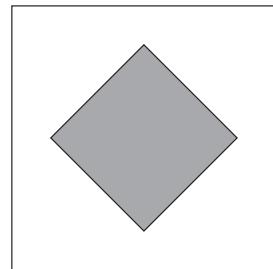
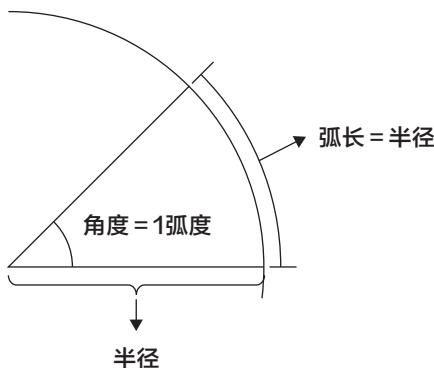
图 13.5



这样的角度是相当直观的。例如，在图13.6中举行围绕其中心旋转45°。

但是，在Processing中，角度同意被指定用弧度代替。一个弧度是圆圈角度的测量，一个弧度圆圈半径与圆的弧长度的比例。180° 为PI弧度，360° 为2*PI弧度，90° 为PI/2的弧度，以此类推。

图 13.7



从度到弧度的转换公式为：

$$\text{弧度} = 2 * \text{PI} * (\text{角度}/360)$$

幸运的是我们更倾向于角度而不是度，Processing会让这个更简单。`radians()`函数能自动的将角度值转换成弧度。此外，常数PI和TWO_PI能够方便的访问相同的角度数字(相当于180° 和360°)。例如下面的代码，将一个形状旋转60° (旋转将在下一章节深入探索)。

```
float angle = radians(60);
rotate(angle);
```

PI是什么

数学常数PI(或者说π)是一个实数，它定义了圆的周长和直径之间的比例。周长约等于直径的3.14159倍。

 练习13-4：一个舞者原地转了2圈。他转了多少度？又转了多少弧度？

Degrees: _____ Radians: _____

13.8 三角函数

三角函数，这个词足够奇怪看起来毫无意义。但是它是计算机进行大量图形计算的基础。任何时候当你需要计算一个角度，确定2点之间的距离，处理圆，弧，线等等问题，你都需要了解三角函数这个基本知识。

三角函数的学习就是研究三角形边和角之间的关系，你需要记住sine，cosine，以及tangent。如图13.8所示。

- soh:sine=对边/斜边
- cah:cosine=邻边/斜边
- toa:tangent=对边/邻边

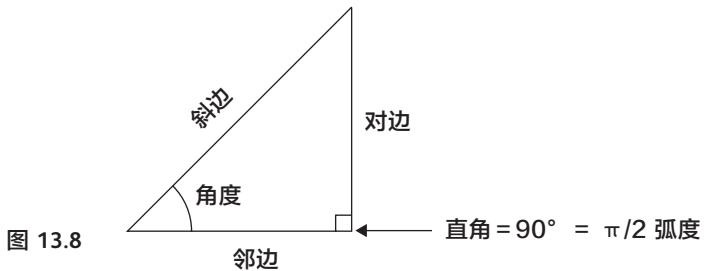


图 13.8

在任何时候，Processing中的图形都需要指定像素点的位置，给予它x和y的坐标。这些坐标被称为直角坐标系，由法国数学家René Descartes命名。

还有另一种有用的坐标系统，被称为极坐标，用从原点的角度以及半径来描述一个点的位置。在Processing中我们不能使用极坐标作为函数的参数。然而，三角函数公式能够允许我们将这些坐标转化成直角坐标。这样就可以用来绘制形状。如图13.9所示。

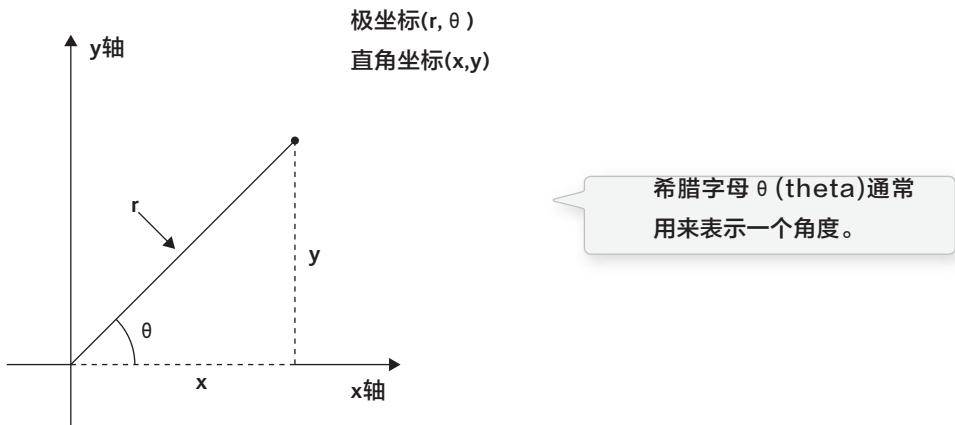


图 13.9

$$\begin{aligned}\sin(\theta) &= y/r \rightarrow y = r * \sin(\theta) \\ \cos(\theta) &= x/r \rightarrow x = r * \cos(\theta)\end{aligned}$$

例如，如果 r 等于 75，角度 θ 等于 45° （或者 $\pi/4$ 弧度），我们可以计算出 x 和 y 的值。正弦 sine 和余弦 cosine 在 Processing 中的函数叫做 `sin()` 和 `cos()`。它们都只包含一个参数，一个基于弧度测量的浮点角度。

```
float r = 75;
float theta = PI / 4; // We could also say: float theta = radians(45);
float x = r * cos(theta);
float y = r * sin(theta);
```

这种类型的转换在某些应用中可能会非常有用。例如，如果使用直角坐标系沿着圆描边？这会非常艰难。但是如果将它变成极点坐标系，这个任务就非常简单了。只需要进行角度的增减。

下面的例子告诉我们怎么使用全局变量以及 `theta`。

例 13-4：极点坐标转换成直角坐标

```
// A Polar coordinate
float r = 75;
float theta = 0;

void setup() {
    size(200,200);
    background(255);
    smooth();
}

void draw() {
    // Polar to Cartesian conversion
    float x = r * cos(theta);
    float y = r * sin(theta);
```

将极点坐标($r, theta$)转换成直角坐标(x, y)用于`ellipse()`函数

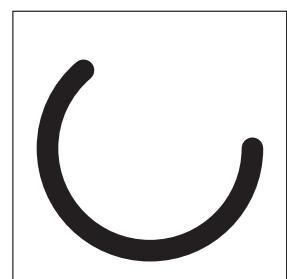
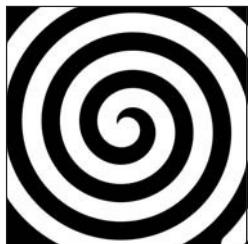


图 13.10

```
// Draw an ellipse at x,y
noStroke();
fill(0);
ellipse(x+width/2, y+height/2, 16, 16); // Adjust for center of window
// Increment the angle
theta += 0.01;
}
```

练习13-5：参考例13-5，画一个由内到外的螺旋形状。请注意这里只需要改变一行代码并添加一行代码即可。



13.9 波动

三角函数可用于更多的几何计算。让我们来看看图13.11，一个sine平面函数 $y=\sin(x)$ 。

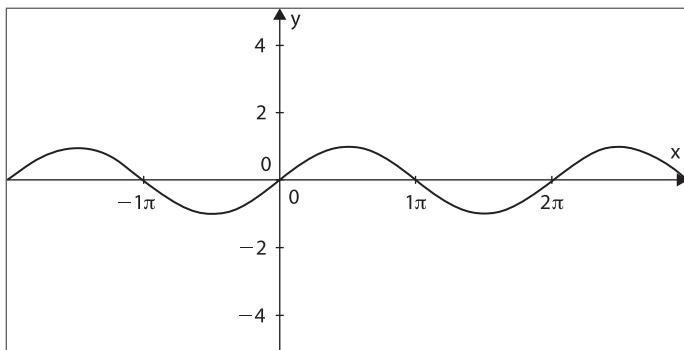


图 13.11

你可能会注意到输出的结果是一条从-1到1之间平滑的曲线。这种类型的行为我们称之为波动，两点之间周期性的运动。就像摆钟。

我们可以在Processing中利用sine函数来分配对象的位置模拟波动。这和我们使用noise()函数控制圆的尺寸类似(详见例13-4)，我们只需要sin()来控制位置。注意noise()产生的是0到1.0之间的数，而sin()产生的是-1到1之间的数。例13-6显示了这个波动代码。

例 13-6：波动

```
float theta = 0.0;

void setup() {
    size(200,200);
    smooth();
}

void draw() {
    background(255);
    // Get the result of the sine function
    // Scale so that values oscillate between 0 and width
    float x = (sin(theta) + 1) * width/2;
    // With each cycle, increment theta
    theta += 0.05;
    // Draw the ellipse at the value produced by sine
    fill(0);
    stroke(0);
    line(width/2,0,x,height/2);
    ellipse(x, height/2,16,16);
}
```

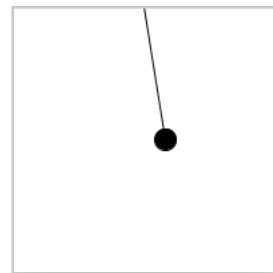


图 13.12

sin()函数的波动显示它在-1到1之间。通过加1我们可以得到0到2之间的值，再通过乘以100，我们可以得到0到200之间的值，它就可以用来作为我们圆的x坐标。

练习13-5：将上面的函数封装到一个波动的对象中。创建一组波动，每一个都会沿着x和y坐标移动。这里有一些构件波动的类的相关代码，能够帮助你开头。



```
class Oscillator {
    float xtheta;
    float ytheta;

    Oscillator() {
        xtheta = 0;
        ytheta = 0;
    }

    void oscillate() {
    }

    void display() {
        float x = _____
        float y = _____
        ellipse(x,y,16,16);
    }
}
```

练习13-7：使用正弦函数 `sin()` 创建一个会“呼吸”的形状，一个会波动的形状。



通过绘制沿路径的正弦函数序列的形状，我们也可以生产一些有趣的结果。参见例13-7。

例 13-7：波动

```
// Starting angle
float theta = 0.0;
void setup() {
    size(200,200);
    smooth();
}

void draw() {
    background(255);
    // Increment theta (try different values for "angular velocity" here)
    theta += 0.02;
    noStroke();
    fill(0);
    float x = theta;
    // A simple way to draw the wave with an ellipse at each location
    for (int i = 0; i <= 20; i++) {
        // Calculate y value based off of sine function
        float y = sin(x)*height/2;
        // Draw an ellipse
        ellipse(i*10,y + height/2,16,16);
        // Move along x-axis
        x += 0.2;
    }
}
```



练习13-8：使用`noise()`函数代替`sin()`重写上面的例子。

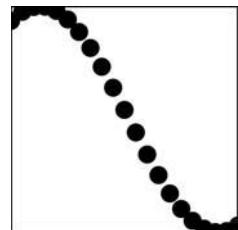


图 13.13

使用for循环来绘制所有波动的点。

13.10 递推

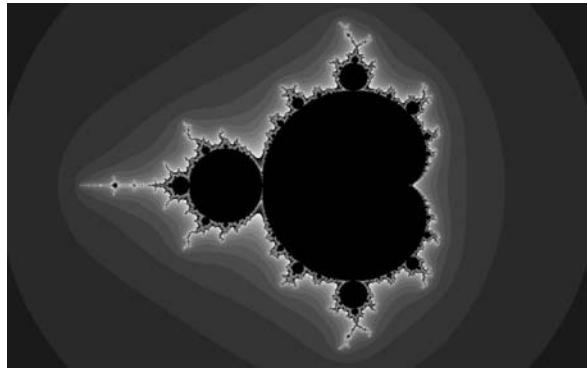


图 13.14

<http://processing.org/examples/manelbrot.html>

在1975年，Benoit Mandelbrot创造了这个词来形容在自然界中相似的形状。很多的东西在我们的物理世界中会遇到理想化的几何形式——信片是长方形的，乒乓球是圆形的，等等。但是，许多自然发生的形状很难用这种简单的描述来概括。如雪花，树木，海岸线以及山。分形提供了一个对这些相似形状的描述和模拟(不管放大或缩小，这些形状最终都是相同的)。生成这些形状的过程被称为递推。

我们知道一个函数可以调用另一个函数。无论我们调用任何函数它都会出现在draw()函数中。但是能否调用函数本身呢？draw()函数能够调用draw()函数吗？实际上是可以的(虽然它实际上是一个糟糕的例子，因为它会导致无限循环)。

函数调用自身叫递推，并且它能够解决不同类型的问题。在数学计算中，这种最常见的例子就是“因子”。

任意数的因子，通常写成 $n!$ ，定义如下：

$$\begin{aligned} n! &= n * n-1 * \dots * 3 * 2 * 1 \\ 0! &= 1 \end{aligned}$$

在Processing中我们可以利用for循环来写一个函数计算因子：

```
int factorial(int n) {
    int f = 1;
    for (int i = 0; i < n; i++) {
        f = f * (i+1);
    }
    return f;
}
```

如果你去仔细观察因子工作，你就会发现一些有趣的事情。让我们看看例 $4!$ 和 $3!$ 。

$$\begin{aligned} 4! &= 4 * 3 * 2 * 1 \\ 3! &= 3 * 2 * 1 \text{ 因此 } ... 4! = 4 * 3! \end{aligned}$$

我们可以用更通俗的语言来形容这些，对于任意正整数 n ：

$$\begin{aligned} n! &= n * (n - 1)! \\ 1! &= 1 \end{aligned}$$

用书面表达就是：

n 的递推被定义为 n 的数量乘以 $n-1$

定义因子中包含因子呢？这个自我参照概念被在函数中被称为递推。我们可以利用递推写一个调用因子本身的函数。

```
int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

很疯狂我知道，但是factorial(4)时，它的工作的原理就如图13.15的过程所示。

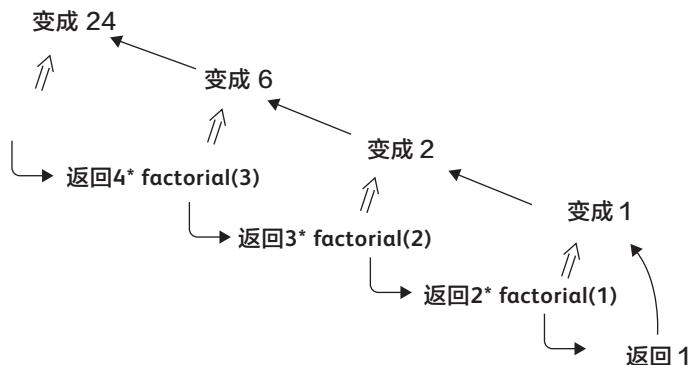


图 13.15

这个有趣的结果同样也可以应用于图形。下面看看递推的函数。如图13.16所示。

```
void drawCircle(int x, int y, float radius) {
    ellipse(x, y, radius, radius);
    if(radius > 2) {
        radius *= 0.75f;
        drawCircle(x, y, radius);
    }
}
```

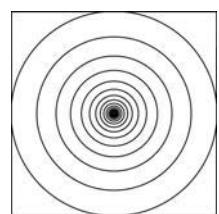


图 13.16

什么是`drawCircle()`在做的呢？它根据收到的参数绘制出圆的形状，并且它可以调用它本身作为相同的参数。其结果就是每一个圆都循环上一个圆的方式。

请注意上面的函数如果半径大于2只会调用本身进行递推。这是一个关键点。所有的递推函数必须有一个推出条件。这是相同的迭代。在第6章中，我们已经了解到所有的for和while循环必须包括一个boolean测试来计算最终结果为false，从而推出循环。如果没有，那么程序会崩溃，陷入一个无限循环的死胡同中。

前面的圆的例子非常简单，因为它完全可以通过简单的迭代实现。但是，在更复杂的情况下，函数调用本身不止一次，递推会变得更有意思。

让我们修改一下drawCircle()让它更复杂一点。对于每一个显示的圆，在它的左边和右边各画一个一半大小的圆。如例13-8所示。

例 13-8：递推

```
void setup() {
    size(200,200);
    smooth();
}

void draw() {
    background(255);
    stroke(0);
    noFill();
    drawCircle(width/2,height/2,100);
}

void drawCircle(float x, float y, float radius) {
    ellipse(x, y, radius, radius);
    if(radius > 2) {
        drawCircle(x + radius/2, y, radius/2);
        drawCircle(x - radius/2, y, radius/2);
    }
}
```

drawCircle()被自身调用了2次，创建了分裂效果。对于每一个圆，都有一个左边和右边小圆。

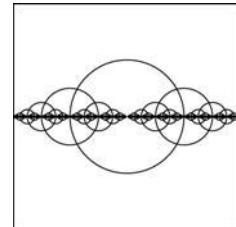


图 13.17

我们可以让圆更小一点，在圆的上方和下方同时再添加圆圈。结果如图13.18所示。

```
void drawCircle(float x, float y, float radius) {
    ellipse(x, y, radius, radius);
    if(radius > 8) {
        drawCircle(x + radius/2, y, radius/2);
        drawCircle(x - radius/2, y, radius/2);
        drawCircle(x, y + radius/2, radius/2);
        drawCircle(x, y - radius/2, radius/2);
    }
}
```

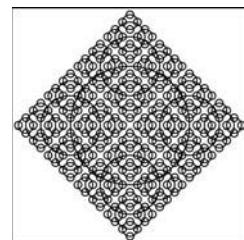
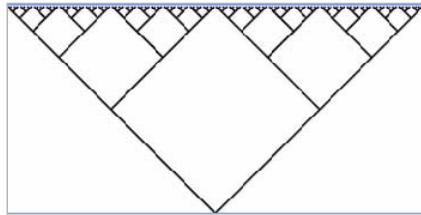


图 13.18

如果这能用迭代代替递推，我就服了你。

练习13-9：完成下面图中图案的代码(注：最好使用线条，虽然它还可以利用旋转的矩形，我们会在第14章中告诉你如何去做)。



```
void setup() {
    size(400,200);
}
void draw() {
    background(255);
    stroke(0);
    branch(width/2,height,100);
}
void branch(float x, float y, float h) {
    _____;
    _____;
    if (_____){
        _____;
        _____;
    }
}
```

13.9 二维数组

在第9章中我们学习到如何用数组跟踪一系列的线性信息，那就是一维数组。但是，与目标系统相关的数据(一个数字图像，一个棋盘游戏等等)都存在与二维中。为了实现可视化这些数据，我们需要一个多维数据结构，也就是一个二维数组。

一个二维数组没有比一个数组多(一个三位数组就是一个数组中的数组中的数组)。想想你的晚餐，你可能有一个关于你吃的东西的一维列表。

(生菜，西红柿，沙拉酱，牛排，土豆泥，四季豆，蛋糕，冰淇淋，咖啡)

或者你会有三个菜的二维列表，每一个都包含三个你吃的东西：

(生菜，西红柿，沙拉酱)和(牛排，土豆泥，四季豆)和(蛋糕，冰淇淋，咖啡)

在使用数组的情况下，我们的老式一维数组看起来就像这样：

```
int[] myArray = {0,1,2,3};
```

一个二维数组看起来就像这样：

```
int[][] myArray = {{0,1,2,3},{3,2,1,0},{3,5,6,1},{3,8,3,4}};
```

对我们而言，我们最好把二维数组看作为一个矩阵。一个矩阵可以通过数组网格安排行和列，有点像宾果游戏。我们下面所写的二维数组阐述了这一点。

```
int[][] myArray = { {0, 1, 2, 3},  
                   {3, 2, 1, 0},  
                   {3, 5, 6, 1},  
                   {3, 8, 3, 4} };
```

我们可以使用这种类型的数据结构对图像进行编码。例如，图13.19的灰阶图像可以用数组表示为：

```
int[][] myArray = { {236, 189, 189, 0},  
                   {236, 80, 189, 189},  
                   {236, 0, 189, 80},  
                   {236, 189, 189, 80} };
```

要贯穿数组中的每一个元素，我们需要使用for循环：

```
int[] myArray = new int[10];  
for (int i = 0; i < myArray.length; i++) {  
    myArray[i] = 0;  
}
```

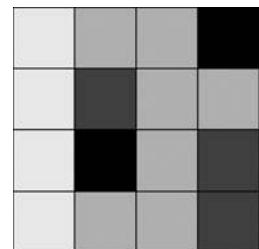


图 13.19

对于一个二维数组，为了引用每一个元素，我们必须使用两个嵌套式循环。这样就能在每一行每一列都有变量的计数。详见图13.20。

```
int cols = 10;  
int rows = 10;  
int[][] myArray = new int[cols][rows];  
  
for (int i = 0; i < cols; i++) {  
    for (int j = 0; j < rows; j++) {  
        myArray[i][j] = 0;  
    }  
}
```

2个嵌套云环允许我们访问二维数组中的每一个点。对于每一个i，都有j相对应。

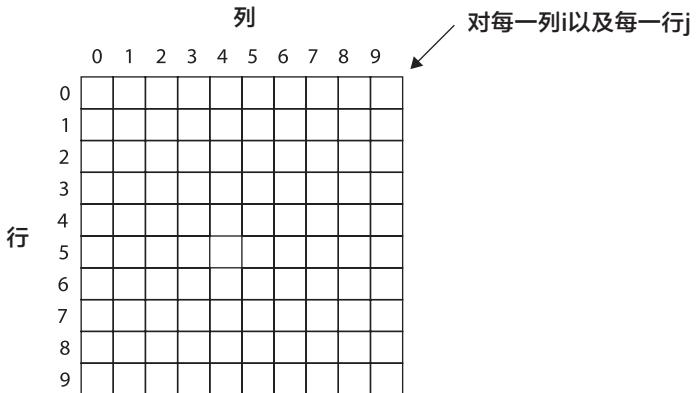


图 13.20

例如，我们可能会使用一个二维数组去编写一个关于绘制图像灰阶的程序，如例13-9所示。

例 13-9：二维数组

```
// Set up dimensions
size(200,200);
int cols = width;
int rows = height;
// Declare 2D array
int[][] myArray = new int[cols][rows];

// Initialize 2D array values
for (int i = 0; i < cols; i++) {
    for (int j = 0; j < rows; j++) {
        myArray[i][j] = int(random(255));
    }
}

// Draw points
for (int i = 0; i < cols; i++) {
    for (int j = 0; j < rows; j++) {
        stroke(myArray[i][j]);
        point(i,j);
    }
}
```

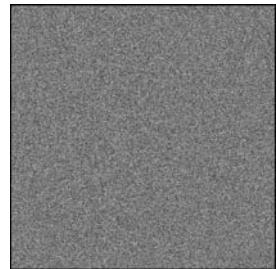


图 13.21

二维数组同样也可以用来储存对象，在编程中涉及“网格”或“板面”相关的东西时，它会很方便使用。例13-10显示了一个储存Cell对象的二维数组网格。每一个cell都是一个矩形，它的亮度由其正弦函数决定，范围从0到255。

例 13-10：包含对象的二维数组

```
// 2D Array of objects
Cell[][] grid;
// Number of columns and rows in the grid
int cols = 10;
int rows = 10;

void setup() {
    size(200,200);
    grid = new Cell[cols][rows];
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            // Initialize each object
            grid[i][j] = new Cell(i*20,j*20,20,20,i + j);
        }
    }
}

void draw() {
    background(0);
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            // Oscillate and display each object
            grid[i][j].oscillate();
            grid[i][j].display();
        }
    }
}

// A Cell object
class Cell {
    float x,y; // x,y location
    float w,h; // width and height
    float angle; // angle for oscillating brightness
    // Cell Constructor

    Cell(float tempX, float tempY, float tempW, float tempH, float tempAngle) {
        x = tempX;
        y = tempY;
        w = tempW;
        h = tempH;
        angle = tempAngle;
    }

    // Oscillation means increase angle
    void oscillate() {
        angle += 0.02;
    }

    void display() {
        stroke(255);
        // Color calculated using sine wave
        fill(127 + 127*sin(angle));
        rect(x,y,w,h);
    }
}
```

二维数组可以用来储存对象。

计数器变量i以及j同样也是列和行的
号码，并且在网格中作为每一个对象
函数构造器中的参数。

cell对象知道关于它在网格中的位
置，同样也知道它的尺寸的变量
x,y,w,h。

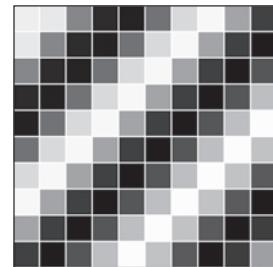


图 13.22

练习13-10：开发一个 Tic-Tac-Toe 游戏。创建一个 Cell 对象，让它存在于两种状态之一：O或者什么都不是。当你点击 cell，他的状态会从什么都没有变成“O”，下面是让你快速开始的框架。

```
Cell[][] board;
int cols = 3;
int rows = 3;

void setup() {
    // FILL IN
}

void draw() {
    background(0);
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            board[i][j].display();
        }
    }
}

void mousePressed() {
    // FILL IN
}

// A Cell object
class Cell {
    float x,y;
    float w,h;
    int state;

    // Cell Constructor
    Cell(float tempX, float tempY, float tempW, float tempH) {
        // FILL IN
    }

    void click(int mx, int my) {
        // FILL IN
    }

    void display() {
        // FILL IN
    }
}
```



练习13-11：如果你感觉没问题，并继续完成这个游戏编程，给它添加一个X，让弯角能够利用点击鼠标交换玩家。

14. 平移和旋转(在3D!)

“什么是矩阵？”

— Neo

本章内容：

- 2D 和 3D 的平移
- 使用 P3D 和 OPENGL
- 顶点形状
- 2D 和 3D 的旋转
- 保存转形后的状态：pushMatrix() 和 popMatrix()

14.1 Z 坐标轴

正如我们通过这本书所看到的，在二维窗口的像素中被称作笛卡尔坐标系：X(水平)和Y(垂直)点。这些概念可以追溯到第1章，当我们开始思考将平面划分为网格。

在三位空间(如实际的，现实的空间)，由提三个轴线(通常被称为Z轴)，它指的是坐标的深度。在Processing草图中，一个坐标沿着Z轴表面它离你窗口坐标向前或向后多少像素。我知道你会很混乱，毕竟一台电脑的窗口也只是二维的。

实际上我们可以创造一个三维的错觉。例如，如果我们向在窗口中绘制一个矩形，并漫漫增加其宽度和高度，它可能看起来就向正朝着你在。如例14-1所示。

例 14-1：一个增长的矩形，或者是一个向你靠近的矩形？

```
float r = 8;

void setup() {
    size(200,200);
}
void draw() {
    background(255);
    // Display a rectangle in the middle of the screen
    stroke(0);
    fill(175);
    rectMode(CENTER);
    rect(width/2,height/2,r,r);
    // Increase the rectangle size
    r++;
}
```

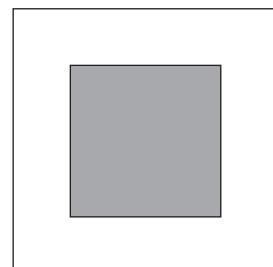


图 14.1

这个矩形会飞出电脑碰到你的鼻子上来吗？从技术上讲，这当然不可能。这仅仅是一个放大的矩形。但是我们已经创造了一个让它向你靠近的幻觉。

幸运的是，对我们来说，如果我们使用3D坐标，Processing会帮助我们创建这个视错觉。虽然在一台电脑显示器上现实第三个维度似乎是虚构的，但是对于Processing来说这是真实的。Processing知道透视以及选择相应的二维像素去创建三维效果。我们应该承认，只要我们进入了3D坐标，一些对于Processing渲染的相关控制必须舍去。你可以不再考虑像素的精确位置，因为在3D视角里，平面形状的X,Y坐标会改变。

为了在三位世界中指定点的位置，坐标系应该按照:x,y,z的顺序。笛卡尔3D系统能够被称作”左撇子”或者”右撇子”。如果你使用右手食指指向y轴正方向(向上),你的拇指指向x轴正方向(向右)，那么其余的手指方向就会指向z轴。如果你用右手按照同样的方式来做，那么它即使左撇子模式。在Processing中，这个系统属于左撇子模式，如图14.2所示。

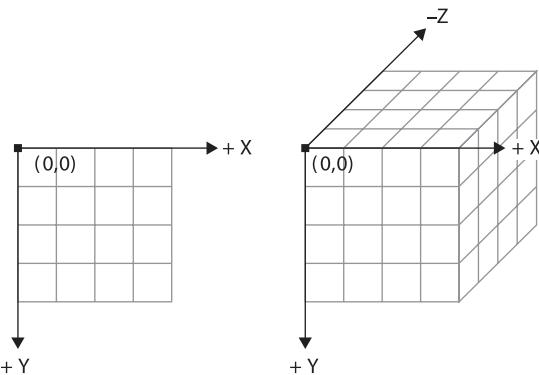


图 14.2

我们首先的目标就是用Processing的3D能力重写例14-1。假设以下变量：

```
int x = width/2;
int y = height/2;
int z = 0;
int r = 10;
```

为了给矩形指定一个坐标，rect()函数需要四个参数：一个x坐标，一个y坐标，宽度，高度。

```
rect(x,y,w,h);
```

我们的第一反应就是给rect()函数添加另一个参数。

`rect(x,y,z,w,h);`

错了！我们不能在 Processing 中使用 (x,y,z) 坐标。在 Processing 中有另外一个函数能够构建 x,y,z 我们会在之后的章节学习到。

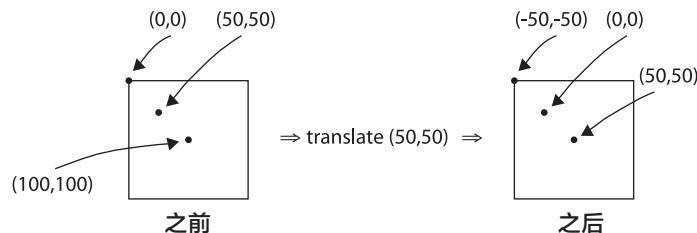
在 Processing reference 中，对于rect()没有这种3种坐标在一起的表达方式。为了给 Processing 中的形状指定3D坐标，我们必须学习一个新函数，它叫作 translate()。

translate() 函数不是唯一的转换3D草图的方式，让我们返回二维环境中他们是如何工作的。

translate() 移动到原点(0,0)，回到之前的状态。我们知道当我们第一个草图开始的时候，原点在窗口的左上角。如果我们调用函数translate()并且给予参数(50,50)，结果如图14.3所示。

原点在哪

在 Processing 草图中”原点”在二维空间的坐标为(0,0)，三维空间的坐标为(0,0,0)。它始终在窗口的左上角，除非你使用 translate() 改变原点坐标。



你可以把它看作是一个在屏幕上笔，这个笔点就代表着原点。

此外，所有的的原点都会从 draw() 重新复位。任何调用translate()指适用与当前周期的draw()循环。如图14-2。

例 14-2：多个转换

```

void setup() {
  size(200,200);
  smooth();
}

void draw() {
  background(255);
  stroke(0);
  fill(175);

  // Grab mouse coordinates, constrained to window
  int mx = constrain(mouseX,0,width);
  int my = constrain(mouseY,0,height);

  // Translate to the mouse location
  translate(mx,my);
  ellipse(0,0,8,8);

  // Translate 100 pixels to the right
  translate(100,0);
  ellipse(0,0,8,8);

  // Translate 100 pixels down
  translate(0,100);
  ellipse(0,0,8,8);

  // Translate 100 pixels left
  translate(-100,0);
  ellipse(0,0,8,8);
}

```

现在我们知道translate()是怎么工作的了，我们就可以返回原点问题来指定3D坐标。不像rect(), ellipse()以及其他函数，它能够给Z坐标使用第三个参数。

```

// Translation along the z-axis
translate(0,0,50);
rectMode(CENTER);
rect(100,100,8,8);

```

上面的坐标轴沿Z轴方向转换50个像素，并且绘制一个(100,100)的矩形。虽然上面的代码从技术上说是正确的，但是将(x,y)指定为转换坐标的一部分是一个很好的习惯，如下所示：

```

// Translation along the z-axis II
translate(100,100,50);
rectMode(CENTER);
rect(0,0,8,8);

```

当我们使用translate()时，矩形的原点坐标会从(0,0)转换成我们需要它去到的坐标。

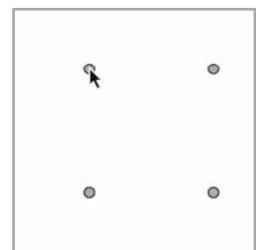


图 14.4

最后我们可以给Z轴上使用一个变量，让这个形状走向我们。

例 14-3：一个向前移动的 Z 坐标

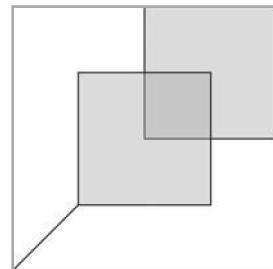
```
float z = 0; // a variable for the Z (depth) coordinate
void setup() {
size(200,200,P3D);
}
void draw() {
background(0);
stroke(255);
fill(100);
// Translate to a point before displaying a shape there
translate(width/2,height/2,z);
rectMode(CENTER);
rect(0,0,8,8);
z++; // Increment Z (i.e. move the shape toward the viewer)
}
```

当我们使用 (x,y,x) 坐标系时，我们必须告诉 Processing 我们想要做一个 3D 草图。在这里我们就需要给 `size()` 函数添加第三个参数“P3D”，详见第 14.2。

虽然看起来和例 14-1 没有什么不同，但是它们是两个完全不同的概念，我们已经开启了三维的大门，这样我们就能利用 Processing 的 3D 引擎在屏幕上作出三维效果了。

 练习 14-1：填写合适的 `translate()` 函数创建这个图案。如果你填写完了，试一试在 `translate()` 中添加第三个参数，将这个图案转化成三维图案。

```
size(200,200);
background(0);
stroke(255);
fill(255,100);
translate(_____,_____);
rect(0,0,100,100);
translate(_____,_____);
rect(0,0,100,100);
translate(_____,_____);
line(0,0,-50,50);
```



当你在绘制一个形状的几何的时候 `translate()` 可以给他们定一个中心点，这是非常有用的。回溯到第十章中我们可以看到这样的代码：

```
void display() {
// Draw Zoog's body
fill(150);
rect( x,y ,w/6,h*2);
// Draw Zoog's head
fill(255);
ellipse( x,y-h/2 ,w,h);
}
```

display()函数绘制了Zoog的所有部分(身体以及头等等), 并且是相对于Zoog的x, y坐标。x,y坐标会呗用于rect() 和 ellipse()。translate() 允许我们将Processing的原点设置为Zoog的(x,y)坐标上, 这样, 其他相关的形状都可以相对于(0,0)来进行绘制。

```
void display() {
    // Move origin (0,0) to (x,y)
    translate(x,y);
    // Draw Zoog's body
    fill(150);
    rect( 0.0 ,w/6,h*2);
    // Draw Zoog's head
    fill(255);
    ellipse(0,-h/2,w,h);
}
```

translate()用于给一个形状集合定义一个原点。

14.2 P3D vs. OPENGL

如果你仔细观察例14-3, 你会发现我们已经在size()函数中添加了第三个参数。传统上来说, size()有一个目的: 定义Processing窗口的大小。size()函数也是可以指定第三个参数告诉Processing绘制模式。它会告诉Processing在显示窗口时用什么方式对图像进行渲染。默认情况下(没有指定任何渲染时)是”JAVA2D”模式, 它会采用现有的JAVA 2D库对图形进行绘制, 上色等等。我们不需要去了解它是怎么工作的。这是Processing开发者需要去做的事。

如果我们想采用3D转换(或者旋转, 我们将在之后的章节中看见), JAVA 2D就不能做到了。在运行时如果依然采用默认模式, 则会出现以下结果:

“translate(x, y, z) can only be used with OPENGL or P3D, use translate(x, y) instead.”

(translate(x, y, z)只能在OPENGL或者P3D模式下使用, 请用translate(x, y)将其代替。)

对于三维模式, 我们希望有不同的模式, 现在我们有2个选择:

P3D—P3D是Processing开发者做创造的一种3D渲染。但是有一点必须指出, 那就是在这种模式下消除锯齿(使用smooth()函数)是不能提供的。

OPENGL—OPENGL是一种采用硬件加速的3D渲染模式。如果在您的计算机中安装了一个兼容OPENGL模式的显卡(几乎每一个电脑都有), 你就可以使用这个模式。虽然在写这本书的时候有一些纠结(你可能会发现P3D和OPENGL只见有些不同), OPENGL在速度方面非常占优势。如果你打算在一个高分辨率屏幕窗口中显示大量的图形, 运用这种模式会展示它的最佳效果。

要指定一个模式, 请在size()函数中添加第三个参数, 并全部大写。

size(200,200);	// 默认的JAVA 2D模式
size(200,200,P3D);	// P3D模式
size(200,200,OPENGL);	//OpenGL模式

当你在使用OPENGL模式的时候，你同样必须倒入OPENGL的库。

你可以通过选择SKETCH, IMPORT LIBRARY 菜单选项来完成这个操作，或者是在草图顶部手动输入以下代码(关于库的说明，详见第12章)：

```
import processing.opengl.*;
```



练习14-2：在Processing草图中分别运用P3D以及OPENGL模式运行一个程序，看看他们有什么不同？

14.3 顶点形状

到现在为止，我们对于图形的绘制能力一直局限于一些简单的二维形状：矩形、椭圆、三角形、线、点。然而对于有一些项目来说，创建一个自定义形状是很有趣的。这个可以通过函数beginShape()和endShape()以及vertex()来完成。

思考一个矩形。在Processing中，矩形呗定义为参考点，宽度和高度。

```
rect(50,50,100,100);
```

但是我们也可以考虑将矩形变成一个多边形(一个封闭的形状)。多边形的点被称为顶点(复数)或顶点(单数)。下面的代码利用了顶点绘制了和rect()函数完全相同的图形。如图14.5所示。

```
beginShape();
vertex(50,50);
vertex(150,50);
vertex(150,150);
vertex(50,150);
endShape(CLOSE);
```

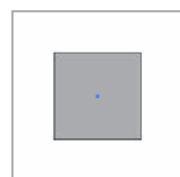


图 14.5

beginShape()表示我们要开始创建一个包含多个顶点的自定义形状：多边形。vertex()指定了多边形的每一个顶点，endShape()表明我们已经完成了顶点的添加。endShape(CLOSE) 中的参数“CLOSE”表明形状应该是闭合的。也就是说最后一个点应该连接到第一个点。

关于使用自定义形状的优点就是它拥有很大的灵活性。例如，你不需要线和线之间是垂直的。如图14.6所示

```
stroke(0);
fill(175);
beginShape();
vertex(50,50);
vertex(150,25);
vertex(150,175);
vertex(25,150);
endShape(CLOSE);
```

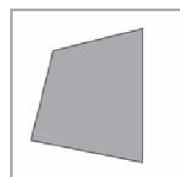


图 14.6

我们同样也可以选择创建多个多边形，利用循环让它有意思起来，如图14.7所示。

```
stroke(0);
for (int i = 0; i < 10; i++) {
    beginShape();
    fill(175);
    vertex(i*20,10-i);
    vertex(i*20 + 15,10 + i);
    vertex(i*20 + 15,180 + i);
    vertex(i*20,180-i);
    endShape(CLOSE);
}
```

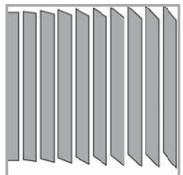


图 14.7

你同样也可以在beginShape()中添加一个参数指定你想创建的形状类型。如果你想使用多个多边形，这个是特别有用的方式。例如，如果你创建了6个顶点，Processing没有办法知道其实你真的是想创建2个三角形(而不是一个六边形)除非你这样写beginShape(TRIANGLES)。如果你不希望创建一个多边形，而是创建一个点或者线，你可以说beginShape(POINTS)或者beginShape(LINES)。如图14.8所示。

```
stroke(0);
beginShape(LINES);
for (int i = 10; i < width; i+=20) {
    vertex(i,10);
    vertex(i,height-10);
}
endShape();
```

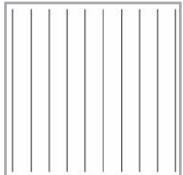


图 14.8

请注意，LINES指的是绘制一些列单独的线，不是一个连续的循环。对于连续的循环，不要使用任何参数。相反，你只需要指定顶点以及noFill()。如图14.9所示。

```
noFill();
stroke(0);
beginShape();
for (int i = 10; i < width; i+=20) {
    vertex(i,10);
    vertex(i,height-10);
}
endShape();
```

在Processing中你可以看到关于beginShape()完整的参数列表。

http://processing.org/reference/beginShape_.html

POINTS, LINES, TRIANGLES, TRIANGLE_FAN, TRIANGLE_STRIP, QUADS, QUAD_STRIP

此外，curveVertex()可以代替vertex()将直线变成曲线。在使用curveVertex()的时候，请注意如何让第一个点和最后一个点不现实。这是因为你需要定义他们的曲线率，开始应设置为第二个点，结束应设置为倒数第二个点。如图14.10所示。

```
noFill();
stroke(0);
beginShape();
for (int i = 10; i < width; i+=20) {
  curveVertex(i,10);
  curveVertex(i,height-10);
}
endShape();
```

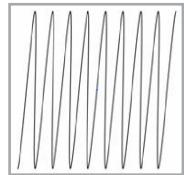
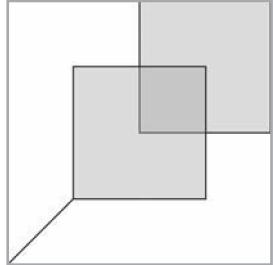


图 14.10

练习14-3：完成下面图片顶点形状的练习


size(200,200);
background(255);
stroke(0);
fill(175);

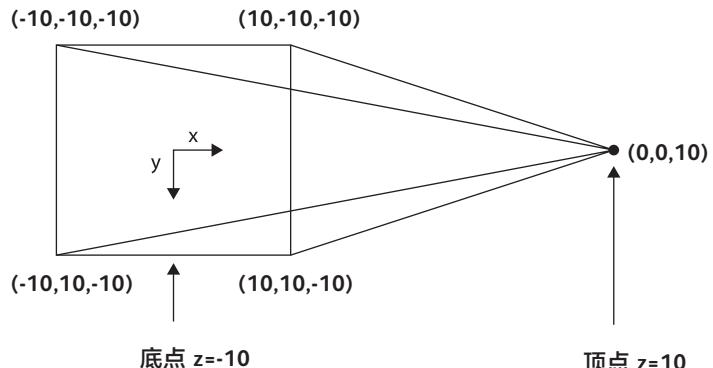
beginShape();
vertex(20, 20);
vertex(_____, _____);
vertex(_____, _____);
vertex(_____, _____);
vertex(_____, _____);
endShape();



14.4 自定义3D形状

三维形状我们同样也可以通过使用beginShape(),endShape()以及vertex()放置多个多边形进行正确的配置。比方说我们想要画一个由4个三角形构成的四面金字塔，所有的的连接都在一个点上(顶部的点)以及一个平面(底面)。如果形状很简单，你可以直接写出其代码。但是在大多数情况下，你最好用铅笔素描草图，确定所有顶点的位置。我们金字塔的例子如图14.11所示。

从图14.11中我们可以看到例14-4的顶点，并且将他们放在一个函数中，这样就允许我们绘制任何尺寸的金字塔了。(作为一个练习，试一试将金字塔转换成一个实际对象。)



```

vertex(-10,-10,-10);    vertex(10,-10,-10);    vertex(10,10,-10);    vertex(-10,10,-10);
vertex(10,-10,-10);    vertex(10,10,-10);    vertex(-10,10,-10);    vertex(-10,-10,-10);
vertex( 0, 0, 10);      vertex( 0, 0, 10);      vertex( 0, 0, 10);      vertex( 0, 0, 10);

```

圖 14.11

例 14-4：一个向前移动的Z坐标

```
void setup() {  
    size(200,200,P3D);  
}
```

```
void draw() {  
    background(255);  
    translate(100,100,0);  
    drawPyramid(150);  
}
```

```
void drawPyramid(int t) {  
    stroke(0);
```

// this pyramid has 4 sides, each drawn as a separate triangle
// each side has 3 vertices, making up a triangle shape
// the parameter "t" determines the size of the pyramid

```
beginShape(TRIANGLES);
fill(255,150);
vertex(-t,-t,-t);
vertex( t,t,-t);
vertex( 0, 0, t);
fill(150,150);
vertex( t,-t,-t);
vertex( t, t,-t);
vertex( 0, 0, t);
fill(255,150);
vertex( t, t,-t);
vertex(-t, t,-t);
vertex( 0, 0, t);
```

一旦金字塔的顶点完成了相关的中心点，我们必须使用translate()将金字塔进行正确的放置。

函数设置顶点围绕着金字塔中心进行灵活的距离变化，这取决于参数传递的数字。

注意每一个多边形都有自己的颜色

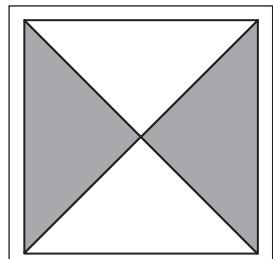


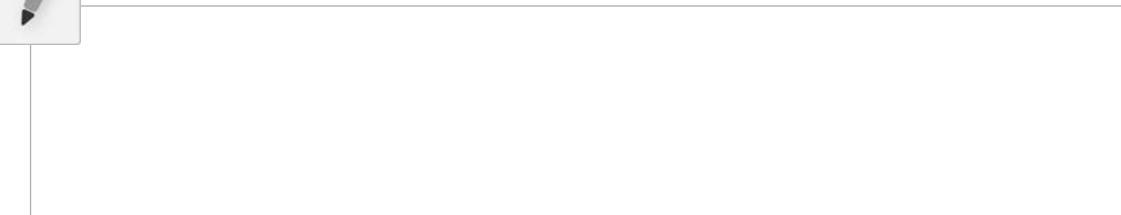
圖 14.12

```

fill(150,150);
vertex(-t, t,-t);
vertex(-t,-t,-t);
vertex( 0, 0, t);
endShape();
}

```

练习14-4 : 创建一个只有3个面的金字塔。包裹底面(合计四个三角形)。使用下面的空间, 勾勒出顶点的位置, 就像图14.11一样。



练习14-5: 使用8个四边形创建一个三维方形, —beginShape(QUADS) (注意在Processing中有一种更简单的方法, 那就是使用box()函数)。

14.5 简单的旋转

这里没有关于金字塔的三维的视觉效果。图像看起来更像是一个扁平的长方形, 并有2条线连接一端到另一端。同样, 我们也必须提醒自己, 我们只是创造了一个三维的错觉。我们将旋转金字塔让它更生动。因此, 让我们来学习旋转。

对于我们来说, 在我们的物理世界中, 旋转是一个很简单很直观的概念。抓住一个接力棒, 转动它, 你就会明白旋转的含义。

很不幸, 编程中的旋转没有想象中那么简单。各种各样的问题都会出现。你应该围绕什么轴旋转? 旋转多少度? 围绕什么原点进行旋转? Processing提供了许多关于旋转的相关函数, 我们需要一步一步慢慢地探索。我们的目标是完成一个太阳系仿真编程, 多个行星围绕恒星以不同的速度进行旋转(以及旋转金字塔, 让它能够有更好的立体感)。

但是, 首先让我们尝试一些简单的操作, 并试图让一个矩形围绕其中心旋转。我们要根据下面三个原则了解旋转:

- 1.在Processing中能够让形状旋转的函数为rotate()。
- 2.rotate()函数有一个参数, 就是旋转角的弧度。
- 3.rotate()旋转形状的方向为顺时针(向右旋转)。

ok，经过简单的了解，我们应该可以使用rotate()函数来进行旋转。让我们尝试 45° （或者说 $\pi/4$ 弧度）的旋转。这是我们的第一次尝试（尽管有缺陷），如图14.13所示。

```
rotate(radians(45));
rectMode(CENTER);
rect(width/2,height/2,100,100);
```

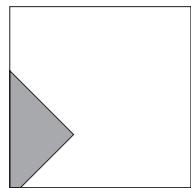


图 14.13

截图，有什么地方出现了问题？矩形看起来确实是旋转了，但是旋转到了错误的位置！

在Processing中实际上有一个最重要的点需要记住的是，形状总是围绕着原点旋转。这个例子中原点在哪？在左上角！这个原点没有被设置，是默认原点的位置。所以矩形没有围绕着自己的中心转，而是围绕着左上角在转，如图14.14所示。

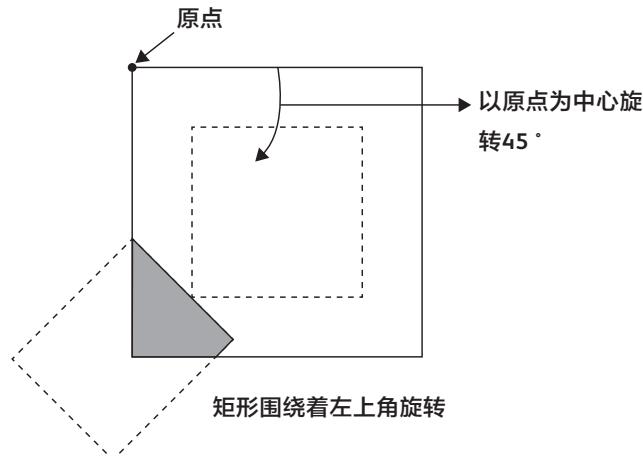


图 14.14

当然，也有可能有一天，你会需要围绕左上角旋转，但是在那一天来临之前，你需要学会将原点移动到正确的位置，并且旋转矩形。`translate()`就是拯救方法。

```
translate(width/2,height/2);
rotate(radians(45));
rectMode(CENTER);
rect(0,0,100,100);
```

我们可以将上面的代码结合`mouseX`的位置进行相关的角度旋转，从而更生动。如例14-5所示。

例 14-5：矩形围绕其中心旋转

```

void setup() {
    size(200,200);
}
void draw() {
    background(255);
    stroke(0);
    fill(175);

    // Translate origin to center
    translate(width/2,height/2);

    // theta is a common name of a variable to store an angle
    float theta = PI*mouseX / width;

    // Rotate by the angle theta
    rotate(theta);

    // Display rectangle with CENTER mode
    rectMode(CENTER);
    rect(0,0,100,100);
}

```

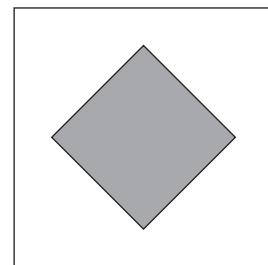
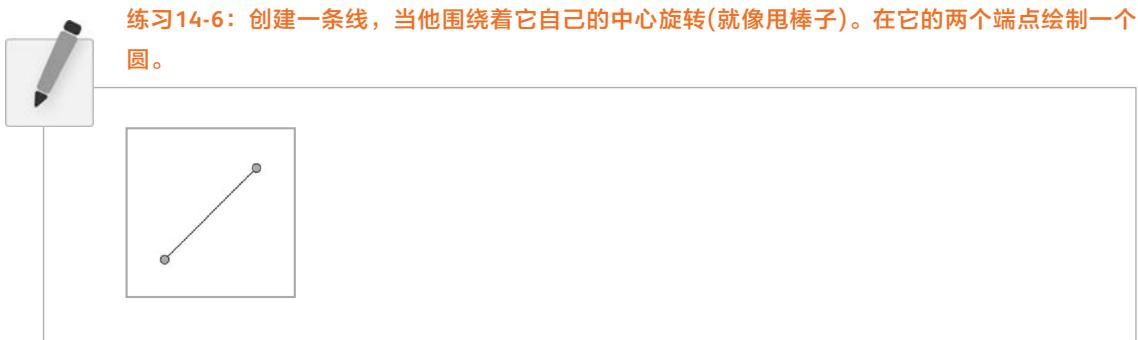


图 14.15

 角度从0到PI，基于草图中的mouseX的坐标
进行改变。



14.6 围绕不同的轴旋转

现在我们已经了解了记本的旋转方式，我们可以对下一个重要环节进行提问了：

我们怎么围绕着轴旋转？

在之前的章节，我们的方形围绕着Z轴旋转。在二维显示中，它是默认的旋转轴。如图14.16所示。

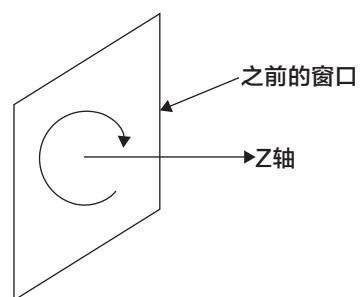


图 14.16

Processing同样也有允许我们绕着x轴旋转或者绕着y轴旋转的函数rotateX()以及rotateY()，但是他们都需要P3D或者OPENGL模式的支持。rotateZ()函数也存在，并且它和rotate()的含义是一样的。详见例14-6, 14-7 和 14-8。

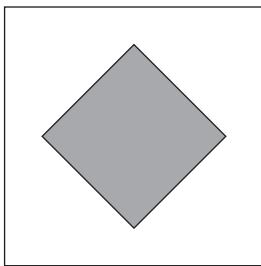


图 14.17

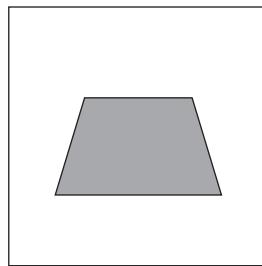


图 14.18

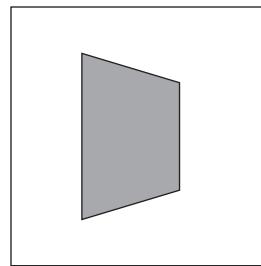


图 14.19

例 14-6：围绕Z轴

```
float theta = 0.0;
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
fill(175);
translate(width/2,
height/2);
rotateZ(theta);
rectMode(CENTER);
rect(0,0,100,100);
theta += 0.02;
}
```

例 14-7：围绕X轴

```
float theta = 0.0;
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
fill(175);
translate(width/2,
height/2);
rotateX(theta);
rectMode(CENTER);
rect(0,0,100,100);
theta += 0.02;
}
```

例 14-8：围绕Y轴

```
float theta = 0.0;
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
fill(175);
translate(width/2,
height/2);
rotateY(theta);
rectMode(CENTER);
rect(0,0,100,100);
theta += 0.02;
}
```

旋转函数同样也可以组合使用。如例14-9 所示，图为14.20

例 14-9：矩形围绕其中心旋转

```
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
fill(175);
translate(width/2,height/2);
rotateX(PI*mouseY/height);
rotateY(PI*mouseX/width);
rectMode(CENTER);
rect(0,0,100,100);
}
```

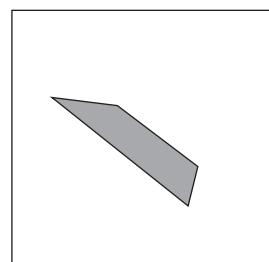


图 14.20

回到金字塔的例子中，我们会看到旋转让三维的感觉有明显的提升。下面的例子同样也扩展到包括了第二个金字塔，他没有使用第一个金字塔的translate。但是请注意在第一个金字塔中旋转围绕的点依然是原点(一旦rotateX()和rotateY()在第二个translate()之前调用出来)。

例 14-10：金字塔

```
float theta=0.0;

void setup(){
    size(200,200,P3D);
}

void draw(){
    background(255);
    theta+=0.01;

    translate(100,100,0);
    rotateX(theta);
    rotateY(theta);
    drawPyraimd(50);

    //再次转换原点
    translate(50,50,20);

    //调用金字塔函数绘制金字塔
    drawPyraimd(10);
}

class drawPyraimd(){
    stroke(0);
    // 这个金字塔有四个面，每一个面都是单独的三角形
    // 每一个面都有三个顶点用来组成三角形
    //参数“t”表示三角形的尺寸

    fill(150,0,0,127);
    beginShape(TRIANGLES);
    vertex(-t,-t,-t);
    vertex( t,-t,-t);
    vertex( 0, 0, t);

    fill(0,150,0,127);
    vertex( t,-t,-t);
    vertex( t, t,-t);
    vertex( 0, 0, t);

    fill(0,0,150,127);
    vertex( t, t,-t);
    vertex(-t, t,-t);
    vertex( 0, 0, t);

    fill(150,0,150,127);
    vertex(-t, t,-t);
    vertex(-t,-t,-t);
    vertex( 0, 0, t);
    endShape(CLOSE);
}
```

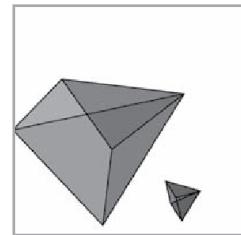


图 14.21



练习14-7：旋转你在练习14-5中创建的3D方形。你能让它围绕着一个角落或者中心旋转吗？你也可以尝试使用函数 box() 创建3D方形。



练习14-8：做一个金字塔的类

14.7 缩放

除了translate()以及rotate()，我们还需要学习一个函数scale()，它能给予形状在屏幕上一些效果。scale()增加或者减少对象在屏幕上的尺寸。正如rotate()，缩放效果也是相对于原来的位置进行执行的。

scale()需要一个浮点数值，并且是按百分比规模来缩放的：1.0等于100%。例如，scale(0.5)就会画出那个对象本身尺寸的50%大小，scale(3.0)就会将对象的尺寸增大到原来的300%大小。

下面是一个重新编写的例14-1(使用scale()增长了矩形的尺寸)。

例 14-11：一个增长的矩形

```
float r = 0.0;
void setup() {
    size(200,200);
}

void draw() {
    background(0);
    // Translate to center of window
    translate(width/2,height/2);
    // Scale any shapes according to value of r
    scale(r);

    // Display a rectangle in the middle of the screen
    stroke(255);
    fill(100);
    rectMode(CENTER);
    rect(0,0,10,10);

    // Increase the scale variable
    r += 0.02;
}
```

scale()使一个对象按照原来的尺寸进行递增。请注意，在增长的同时，它形状的轮廓也随之变粗。

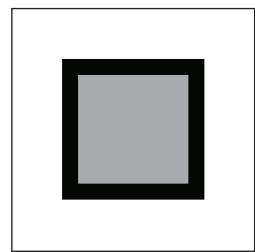


图 14.22

scale()同样也可以使用两个参数(x轴和y轴上沿着不同的值缩放)或者三个参数(x轴，y轴，以及z轴)。

14.8 矩阵：推入和弹出

什么是矩阵？

为了保持旋转和平移的轨道，让形状形状进行不同的变化，Processing需要使用到一个矩阵(几乎所有的电脑平面软件都会使用到这个功能)。

矩阵的变幻已经超出了这本书的范围，但是如果你想让坐标系统的相关信息存储更为简单，你就需要学习转换矩阵。当平移或者旋转的时候，变换矩阵就会改变。它会时不时的存储当前矩阵的状态，并且过一会又会重复储存。这最终会让我们方便移动或旋转这个单独的形状而不影响其他东西。

什么是矩阵？

矩阵是行和列数的一个表。在Processing中，一个变换矩阵经常被用来定义窗口的朝向—平移或者旋转？你可以在任何时间调用printMatrix()函数来查看当前矩阵。这就是矩阵的“正常”状态，不用调用translate()或者rotate()。

```
1.0000  0.0000  0.0000
0.0000  1.0000  0.0000
```

这个概念最好是假设一个例子来说明。让我们给自己一个任务：创建一个Processing草图，让它有2个矩形按照不同的速度和不同的方向围绕着他们各自的中心旋转。

当我们开始开发这个例子时，我们的问题就随之而来，这个时候我们就需要pushMatrix()和popMatirx()函数来实现我们需要的功能。

这和章节14.4基本上用的是相同的代码，我们可以让一个方形绕着z轴左上角进行旋转。如例14-12。

例 14-12：一个旋转的矩形

```
float theta1 = 0;
void setup() {
    size(200,200,P3D);
}
void draw() {
    background (255);
    stroke(0);
    fill(175);
    rectMode(CENTER);
    translate(50,50);
    rotateZ(theta1);
    rect(0,0,60,60);
    theta1 += 0.02;
}
```

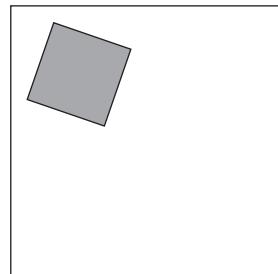


图 14.23

现在做一些调整，我们让方形在底部右下角旋转。

例 14-13：旋转另一个矩形

```
float theta2 = 0;
void setup() {
    size(200,200,P3D);
}

void draw() {
    background (255);
    stroke(0);
    fill(175);
    rectMode(CENTER);
    translate(150,150);
    rotateY(theta2);
    rect(0,0,60,60);
    theta2 += 0.02;
}
```

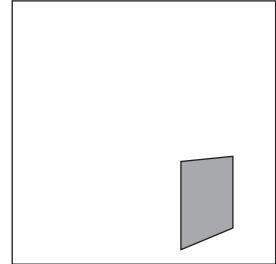


图 14.24

如果不仔细想想，我们可以会直接结合这2个编程内容，`setup()`函数是相同的，我们需要声明2个全局变量，`theta1` 和 `theta2`，并且给每个矩形调用响应的旋转和平移。我们同样需要调整平移的位置，第二个矩形为`translate(150,150)`,第二个矩形根据第一个矩形的位置进行平移`translate(50,50)`，所以我们实际上要填写`translate(100,100)`。

```
float theta1 = 0;
float theta2 = 0;

void setup() {
    size(200,200,P3D);
}

void draw() {
    background(255);
    stroke(0);
    fill(175);
    rectMode(CENTER);
    translate(50,50);
    rotateZ(theta1);
    rect(0,0,60,60);
    theta1 += 0.02;
    translate(100,100);
    rotateY(theta2);
    rect(0,0,60,60);
    theta2 += 0.02;
}
```

首先调用**rotateZ()**对形状进行旋转。而这样，2个矩形都会跟第一个矩形一样的旋转。

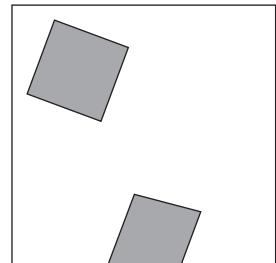


图 14.25

运行这个例子你会很快就发现一个问题。第一个方形(左上角)围绕其中心旋转。但是第二个方形并没有围绕着自己的中心旋转，而是围绕着第一个方形的中心旋转！请记住，所有的平移和旋转都是相对于前一个的坐标系统状态进工作的。我们需要通过矩阵来重置原来的状态，使形状可以单独平移和旋转。

保存和重置平移/旋转状态的函数使pushMatrix()和popMatrix()。在上手之前，让我们假设函数saveMatrix()和restoreMatrix()(注意，这2个假设的函数是不存在的) push = save， pop=restore。

让每个方形围绕自身旋转，我们可以写成下面的算法(粗体部分)。

1.保存当前的变换矩阵。我们开始的原点在窗口左上角的(0,0)，并且没有旋转。

2.平移和旋转第一个矩形。

3.显示第一个矩形。

4.重置回第一步，这样就不会影响第2,3 步。

5.平移和旋转第二个矩形。

6.显示第二个矩形。

我们将代码进行了重写，答案在例14-14中，结果如图14.26所示。

例 14-14：2个分开旋转的矩形

```
float theta1 = 0;
float theta2 = 0;

void setup() {
    size(200,200,P3D);
}

void draw() {
    background(255);
    stroke(0);
    fill(175);
    rectMode(CENTER);

    pushMatrix();
    translate(50,50);
    rotateZ(theta1);
    1
    rect(0,0,60,60);
    popMatrix();
    2

    pushMatrix();
    translate(150,150);
    rotateY(theta2);
    3
    rect(0,0,60,60);
    popMatrix();
    4

    theta1 += 0.02;
    theta2 += 0.02;
}
```

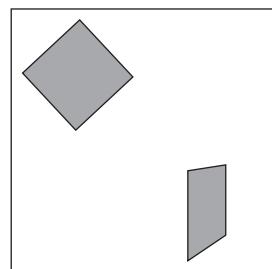


图 14.26

虽然从技术上说不需要第二个矩形前后放置pushMatrix()和popMatrix()，但是这种书写方式是一个好习惯（有些时候我们还有其他的形状需要添加）。在平移或者旋转之前使用pushMatirx()和popMatrix()是一个好的经验法则，这样能够使每个形状单独的运行其动作。实际上这个例子真正的情况应该是面向对象编程，让每一个对象拥有他们自己的pushMatrix()，translate()，rotate()，和 popMatrix()。如例 14-5所示。

例 14-15：让你的对象旋转起来

```
// An array of Rotater objects
Rotater[] rotaters;

void setup() {
    size(200,200);
    rotaters = new Rotater[20];
    // Rotaters are made randomly
    for (int i = 0; i < rotaters.length; i++) {
        rotaters[i] = new Rotater(random(width),random(height),random(-0.1,0.1),random(48));
    }
}

void draw() {
    background(255);
    // All Rotaters spin and are displayed
    for (int i = 0; i < rotaters.length; i++) {
        rotaters[i].spin();
        rotaters[i].display();
    }
}

// A Rotater class
class Rotater {
    float x,y; // x,y location
    float theta; // angle of rotation
    float speed; // speed of rotation
    float w; // size of rectangle

    Rotater(float tempX, float tempY, float tempSpeed, float tempW) {
        x = tempX;
        y = tempY;
        theta = 0; // Angle is always initialized to 0
        speed = tempSpeed;
        w = tempW;
    }

    // Increment angle
    void spin() {
        theta += speed;
    }
}
```

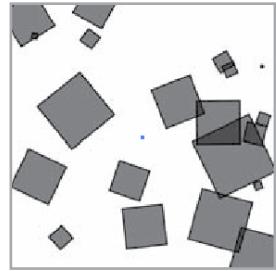


图 14.27

```
// Display rectangle
void display() {
    rectMode(CENTER);
    stroke(0);
    fill(0,100);
    // Note the use of pushMatrix()
    // and popMatrix() inside the object's
    // display method
    pushMatrix();
    translate(x,y);
    rotate(theta);
    rect(0,0,w,w);
    popMatrix();
}
}
```

pushMatrix()和popMatrix()被称为类中display()的函数。通过这种方式，旋转的对象能够各自独立的进行平移和旋转。

有趣的是你也可以通过嵌套多个pushMatrix()和popMatrix()来得到更有意思的结果。这必须同时调用pushMatrix()和popMatrix()函数，但是他们不会一个接着一个。

要了解这个的工作流程，那就让我们来仔细分析一下”push” 和”pop”的意思。”push” 和”pop”是计算机科学中的一个堆栈的概念。了解堆栈的概念能后很好的帮助我们正确学习pushMatrix()和popMatrix()。

思考一下一个英语老师，在备课的时候将文件分级放在桌子上，它们被一摞一摞的叠加放置在桌上。老师读取它们时顺序和排列的时候刚好相反。第一个放在桌上的文件是最后一个读的。最后一个放在桌上的文件编程第一个读的。注意，这是完全相反的顺序。如果你是在排队买电影票，第一个进电影院的人就是第一个去买票的人，最后一个进去的就是最后一个去买票的。如图14.28。

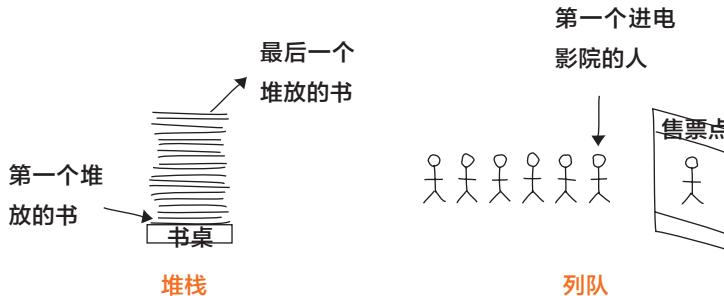


图 14.28

push是堆栈中放入一些东西的过程，pop是堆栈中弹出一些东西的过程。这就是为什么我们必须要有完全相等数量的pushMatrix()和popMatrix()来进行使用。如果某些东西根本不存在，那么你肯定也就弹不出来。(如果你push和pop的数量不正确，Processing就会告诉你”Too many calls to popMatrix() (and not enough to pushMatrix”)

我们以旋转矩形的编程为基础，看看我们是如何进行pushMatrix()和popMatrix()的嵌套。下面的草图右一个圆在中央(我们估且称之为sun)还有一个旋转的圆(称之为earth)，另外还有2个圆围绕着地球转(称之为moon #1和moon #2)。

例 14-16：让你的对象旋转起来

```
// Angle of rotation around sun and planets
float theta = 0;

void setup() {
    size(200,200);
    smooth();
}

void draw() {
    background(255);
    stroke(0);

    // Translate to center of window
    // to draw the sun.
    translate(width/2,height/2);
    fill(255,200,50);
    ellipse(0,0,20,20);

    // The earth rotates around the sun
    pushMatrix();
    rotate(theta);
    translate(50,0);
    fill(50,200,255);
    ellipse(0,0,10,10);

    // Moon #1 rotates around the earth
    pushMatrix();
    rotate(-theta*4);
    translate(15,0);
    fill(50,255,200);
    ellipse(0,0,6,6);
    popMatrix();

    // Moon #2 also rotates around the earth
    pushMatrix();
    rotate(theta*2);
    translate(25,0);
    fill(50,255,200);
    ellipse(0,0,6,6);
    popMatrix();

    popMatrix();
    theta += 0.01;
}
```

pushMatrix()用于保存在绘制 moon #1之前的状态。这样在画moon #2 之前我们就能重置到现在状态。这2个月亮都围绕着地球转(而地球又围绕着太阳转)。

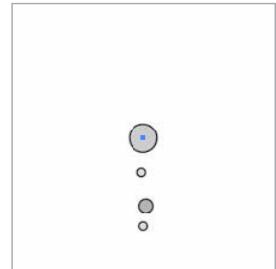


图 14.29

pushMatrix()和**popMatrix()**同样也可以嵌套在**for**循环或**while**循环内，产生更有意思的结果，下面的例子有一点点难，但是我希望你能够去试一试。

例 14-17：嵌套push和pop

```
// Global angle for rotation
float theta = 0;

void setup() {
    size(200, 200);
    smooth();
}

void draw() {
    background(100);
    stroke(255);

    // Translate to center of window
    translate(width/2,height/2);

    // Loop from 0 to 360 degrees (2*PI radians)
    for(float i = 0; i < TWO_PI; i += 0.2) {
        // Push, rotate and draw a line!
        pushMatrix();
        rotate(theta + i);
        line(0,0,100,0);
        // Loop from 0 to 360 degrees (2*PI radians)
        for(float j = 0; j < TWO_PI; j += 0.5) {
            // Push, translate, rotate and draw a line!
            pushMatrix();
            translate(100,0);
            rotate(-theta-j);
            line(0,0,50,0);
            // We're done with the inside loop, pop!
            popMatrix();
        }
        // We're done with the outside loop, pop!
        popMatrix();
    }

    // Increment theta
    theta += 0.01;
}
```

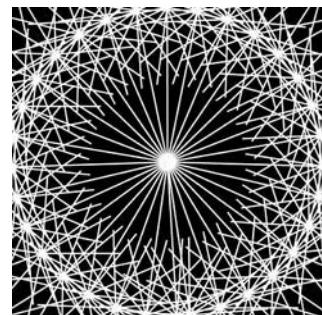


图 14.30

变换的状态已经在每一个圆
通过for循环的时候保存和重
置。请从注释中看到不同。



练习14-9：编写一个金字塔或者方形得类。让它得每个对象都能调用自己得pushMatrix()和popMatrix()。并且试一试看能不能在3D的环境中让它们每个对象独立的旋转？

14.9 太阳系系统

利用本章所学的平移，旋转，push和pop技术，我们准备建立一个太阳系的系统。我们将更新之前章节例14-16的内容，有2个主要改变：

- 每一个星球都是一个对象，它们都是星球类的其中一员。
- 一个数组的星球会围绕着太阳旋转。

例 14-18：面向对象的太阳系

```
// An array of 8 planet objects
Planet[] planets = new Planet[8];

void setup() {
    size(200,200);
    smooth();
    // The planet objects are initialized using the counter variable
    for (int i = 0; i < planets.length; i++) {
        planets[i] = new Planet(20 + i*10,i + 8);
    }
}

void draw() {
    background(255);
    // Drawing the Sun
    pushMatrix();
    translate(width/2,height/2);
    stroke(0);
    fill(255);
    ellipse(0,0,20,20);
    // Drawing all Planets
    for (int i = 0; i < planets.length; i++) {
        planets[i].update();
        planets[i].display();
    }
    popMatrix();
}

class Planet {
    float theta; // Rotation around sun
    float diameter; // Size of planet
    float distance; // Distance from sun
    float orbitspeed; // Orbit speed

    Planet(float distance_, float diameter_) {
        distance = distance_;
        diameter = diameter_;
        theta = 0;
        orbitspeed = random(0.01,0.03);
    }

    void update() {
        // Increment the angle to rotate
        theta+= orbitspeed;
    }

    void display() {
        pushMatrix();
        rotate(theta); // rotate orbit
        translate(distance,0); // translate out distance
        stroke(0);
        fill(175);
        ellipse(0,0,diameter,diameter);
        // After we are done, restore matrix!
        popMatrix();
    }
}
```

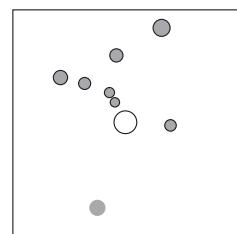


图 14.31

每一个星球都有自己旋转的角度

利用pushMatrix()保存平移
或者旋转之前的状态。

一旦绘制完当前的星球，它的状态
就会用popMatrix()进行重置，
这样就不会影响到下一个星球。



练习14-10：怎样在行星上添加卫星(卫星围绕行星转，行星围绕太阳转)？提示：卫星的类和行星的类基本上一致。但是，会在行星的类中加入一个卫星变量(在第22章中，我们将会学习到更高效和先进的面向对象编程技术)。



练习14-12：将太阳系扩展成三维的系统。尝试使用 `sphere()` 或者 `box()` 来代替 `ellipse()`。注意 `sphere()` 使用一个参数,球的半径。`box()` 使用一个参数(在正方体的情况下)或者三个参数(宽, 高, 深度)。



第六课项目

创建一个虚拟的生态系统。做一个类，让它适应于每个”生物”对象。使用第13到第14章所学的只是，让这个生态系统变的有趣：

- 使用Perlin noise 控制生物的运动。
- 使生物看起来在呼吸。
- 使用递推设计生物。
- 使用beginShape()自定义生物的形状。
- 使用旋转进行生物的行为。

使用下面的草图空间，完成这些练习，并且使用伪代码先进行构想。

第七课

显微镜下的像素

15.图像

16.视频

15. 图像

“当像素出现时，它就填满了我的生活。我可能需要几十年才能消化这些在我手指和脑中的像素。”

— John Maeda

本章内容：

- PImage类
- 显示图像
- 改变图像色彩
- 图像中的像素
- 简单的图像处理
- 交互式的图像处理

数字图片没有其他的数据—它利用红色，绿色和蓝色在一每个像素点上进行颜色填充。在大多数时候，我们在电脑上看到这些像素就是多个小像素加在一起的三明治。然而随着一些创造性的四位和一些简单的像素代码操作，我们可以给边这些像素的显示信息。在这一章节我们就是要打破简单的形状绘制，并且运用图像(以及它们的像素)来生成Processing中的平面。

15.1 图像入门

但是现在，我们还是把思想停留在数据类型。我们通常会指定一些变量，一个浮点变量speed，一个整数变量x，可能还需要一个字符letterGrade。这些都是原始的数据类型。虽然这有一些麻烦，但是我们同样也开始看到了对象之类的复杂数据类型(里面包含函数)—我们的Zoog的类，它包含了坐标，尺寸以及速度的浮点变量以及move,display函数本身等等。当然Zoog是一个用户自定义的类，我们将Zoog带进了编程的世界，并且明确它是什么，并且给予Zoog对象数据和函数。

除了用户自定义的对象，Processing同样还有一些非常方便的类已经准备好了，不需要我们写入任何代码。(在第23章中，我们将会发现我们能够访问Java库中的类)。我们第一个要研究的就是在Processing中已经自带的类PImage，这个类能够读取和显示图片，如图15.1所示。



图 15.1

例 15-1：“Hello World”图像

```
// Declaring a variable of type PImage
PImage img;

void setup() {
    size(320,240);
    // Make a new instance of a PImage by loading an image file
    img = loadImage("mysummervacation.jpg");
}

void draw() {
    background(0);
    image(img,0,0);
}
```

声明一个变量类型 `PImage`, Processing 的核心库中提供这个类。

image()函数显示了图片的位置, 在这里为(0,0)。

使用`PImage`对象实例不像使用其他用户自定义的类。首先, 变量类型是`PImage`, 名称”`img`”被声明。然后, 一个新的`PImage`对象通过`loadImage()`函数被创建。`loadImage()`需要一个参数, 一个字符串`String`(关于字符串的详细探索我们会在第17章中学到)的文件名称, 并加载这个文件到内存中。`loadImage()`会在你Processing草图的数据文件夹中寻找这个图像文件。

数据文件夹: 怎样找到它?

图像可以通过下列方法自动被添加到数据文件夹:

Sketch → Add File...

或者手动添加:

Sketch → Show Sketch Folder

这样就会打开草图文件夹, 如图15.2所示。如果没有数据文件目录, 那么你就创建一个。如果不创建, 那么就将你的图像文件直接放进去即可。Processing接受下列格式的图像文件: GIF, JPG, TGA, 和PNG



图 15.2

在例15-1, 你会发现我们从来没有用过类似”`new PImage()`”的”函数构造器”来实现`PImage`对象的实例。毕竟, 在所有相关的例子来说, 使用函数构造器就会产生一个对象实例。

```
Spaceship ss = new Spaceship();
Flower flr = new Flower(25);
```

然而会有：

```
PImage img = loadImage("file.jpg");
```

实际上，`loadImage()`函数就执行了函数构造器的工作，它会从指定的文件名称中返回一个全新的`PImage`对象实例。我们可以认为它就是`PImage`加载图像文件的函数构造器。对于创建一个空白的图像，请使用`createImage()`函数。

```
// Create a blank image, 200X200 pixels with RGB color
PImage img = createImage(200,200,RGB);
```

我们同样也应该注意从硬盘中向内存加载图片这个过程比较缓慢，我们必须确保我们的程序只在`setup()`中加载一次这样的图片。在`draw()`中加载图片很有可能会导致程序的性能缓慢，并出现“内存不足”的错误。

一旦图片完成加载，我们就需要`image()`函数显示它。`image()`函数必须包括3个参数—需要显示的图片，x坐标，y坐标。或者添加2个参数，让图片调整到一定的宽度和高度。

```
image(img,10,20,90,60);
```



练习15-1：加载并且显示一个图片。利用鼠标的坐标控制图片的宽度和高度。

15.2 图像的动画

在这里，你可以很轻松的看见你能使用图片将之前章节的例子进行进一步开发。

例 15-2：“Hello World”图像

```
PImage head; // A variable for the image file
float x,y; // Variables for image location
float rot; // A variable for image rotation
```

```
void setup() {
    size(200,200);
    // load image, initialize variables
    head = loadImage("face.jpg");
    x = 0.0f;
    y = width/2.0f;
    rot = 0.0f;
}
```

```
void draw() {
    background(255);
    // Translate and rotate
    translate(x,y);
    rotate(rot);
    image(head,0,0); // Draw image
```

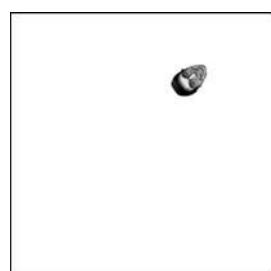


图 15.3

图像就像普通的形状一样可以使用变量,`translate()`,
`rotate()`等等,进行动画效果处理。

```
// Adjust variables to create animation
x += 1.0;
rot += 0.01;
if (x > width) {
    x = 0;
}
}
```

练习15-2：用面向对象的风格重写这个例子，其中有一些图像的坐标，尺寸，旋转等一些数据都包含在这个类中。你能让我们碰到屏幕边缘的时候让图像进行交换吗？



```
class Head {
    _____ // A variable for the image file
    _____ // Variables for image location
    _____ // A variable for image rotation
    Head(String filename, _____, _____) {
        // Load image, initialize variables
        _____ = loadImage(_____);
        _____
        _____
        _____
    }
}

void display() {
    _____
    _____
    _____
}

void move() {
    _____
    _____
    _____
    _____
}
```

字符串同样是一个类，在第17章中我们会进行详细的探索。

15.3 我的第一个 Processing 图像滤镜

对于显示的图像，我们想让它有所改变。或许我们想让图像有明暗的调整，或者是透明度，或者是其他什么效果。这些效果就是简单的图片滤镜，我们可以使用Processing中的tint()函数来完成。tint()本质上说就等于图形中的fill()，它可以设置显示图像的颜色以及透明度。tint()的参数能够简单的指定用于图像中的每一个像素，同样也指定这些像素的透明度。

在下面的例子中，我们将假设2个图像(一个向日葵图像和一条狗图像)已经被加载，并且狗的图像被用于背景(这样就允许我们将透明度显示出来)。如图15.4所示，如果你需要看到彩色版本，请访问<http://www.learningprocessing.com>。

```
PImage sunflower = loadImage("sunflower.jpg");
PImage dog = loadImage("dog.jpg");
background(dog);
```

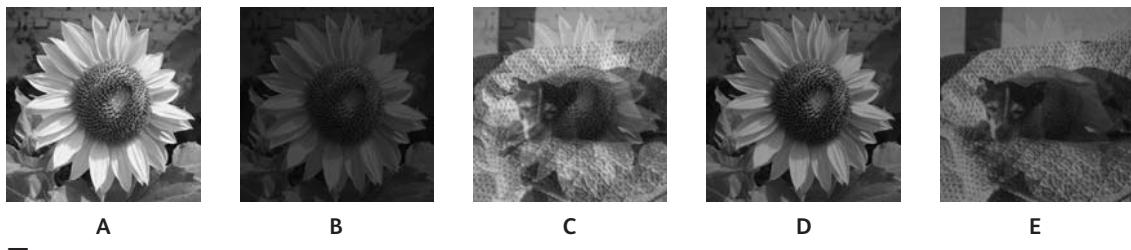


图 15.4

如果tint()中只包含一个参数，那么这只会影响图片的亮度。

```
tint(255);
image(sunflower,0,0);
```

A 图像保留了其原有的状态

```
tint(100);
image(sunflower,0,0);
```

B 图像会变暗

第二个参数会改变图像的透明度。

```
tint(255,127);
image(sunflower,0,0);
```

C 图像改变为50%的不透明度

如果有三个参数会影响其红色，绿色，蓝色的份量。

```
tint(0,200,255)
image(sunflower,0,0);
```

D 图像改变没有红色，大部分的绿色，以及全部的蓝色

最后，添加第四个参数来进行透明度的控制(和2个参数的方法相同)。顺便说，tint()的指定范围值能够被colorMode()指定(详见第1章)。

E 图像改变全部的红色以及部分透明度



练习 15-3：使用 tint() 显示图像。利用鼠标的位置控制图像的红，绿，蓝色调。同样使用鼠标到角落或者中心的距离来控制色调调试一试。



练习15-4：使用tint()创建一个蒙太奇图像。当你有大量的图像层的时候会发生什么，每一个不同的透明度？你能作出互交让不同的图像淡入淡出吗？

15.4 一组数组图片

单张图片是一个美好的开始。但是它对我们来说不会坚持很久，我们会想要去探索多个图片。是的，我们可以跟踪多个变量图片，并且这会是我们重新回顾数组的一个好机会。假设我们有5个图片，并且没当用户点击鼠标时，我们想让背景图片进行更换。

首先，我们需要建立图像的数组，作为一个全局变量。

```
// Image Array
PImage[] images = new PImage[5];
```

其次，我们加载每个图片到数组中合适的位置。这个发生在setup()中。

```
// Loading Images into an Array
images[0] = loadImage("cat.jpg");
images[1] = loadImage("mouse.jpg");
images[2] = loadImage("dog.jpg");
images[3] = loadImage("kangaroo.jpg");
images[4] = loadImage("porcupine.jpg");
```

当然，这样书写有些尴尬。加载每个图像是不是有一些不优雅。这里只有5个图，可以一个一个的写，但是如果100个就非常复杂了。有一种解决方式就是将文件名称储存到字符串数组种，并且使用for循环豫剧对所有的数组元素进行初始化。

```
// Loading Images into an Array from an array of file names
String[] filenames = {"cat.jpg", "mouse.jpg", "dog.jpg", "kangaroo.jpg", "porcupine.jpg"};
for (int i = 0; i < filenames.length; i++) {
    images[i] = loadImage(filenames[i]);
}
```

串连:一种新的方式

通常情况下，我们的加号表明相加， $2+2=4$ ，对吗？

但是在文字(存储在一个字符串例，用引号括起来)情况下，+代表串连，这就是让这2个字符串在一起。

“Happy”+“ Trails”=“Happy Trails”

“2”+“ 2”=“22”

关于更多字符串的介绍，请参考第17章。

更好的是，如果我们花一些时间去重新整理我们的图像名称编号(“animal1.jpg”，“animal2.jpg”，“animal3.jpg”等等)我们就可以真正的简化下面的代码了：

```
// Loading images with numbered files
for (int i = 0; i < images.length; i++) {
    images[i] = loadImage("animal" + i + ".jpg");
}
```

一旦图像加载完毕，我们就要到draw()中。这里我们通过引用索引(“0”如下所示)来显示特定的图像。

```
image(images[0],0,0);
```

当然，使用硬编码来采用索引值是非常愚蠢的。我们需要一个变量让它在不同时间显示不同图片。

```
image(images[imageindex],0,0);
```

“imageindex”变量应该被作为一个全局变量进行声明(一个整数类型)。它的值可以通过整个程序进行改变。完整版如例15-3所示。

例 15-3：切换的图像

```
int maxImages = 10; // Total # of images
int imageIndex = 0; // Initial image to be displayed is the first
PIImage[] images = new PImage[maxImages]; // The image array
```

声明一个图像数组

```
void setup() {
    size(200,200);
    // Loading the images into the array
    // Don't forget to put the JPG files in the data folder!
    for (int i = 0; i < images.length; i++) {
        images[i] = loadImage("animal" + i + ".jpg");
    }
}
```

加载图像数组

```
void draw() {
    image(images[imageIndex],0,0); // Displaying one image
}
```

从数组中选择一个图像

```
void mousePressed() {
    // A new image is picked randomly when the mouse is clicked
    // Note the index to the array must be an integer!
    imageIndex = int(random(images.length));
}
```

鼠标点击时，索引值会随机选择，并变化新的图像

对于图片的切换是按序列的这种动画，请看例15-4。

例 15-4：按顺序切换图像

```
void draw() {
    background(0);
    image(images[imageIndex],0,0);
    // increment image index by one each cycle
    // use modulo "%" to return to 0 once the size
    // of the array is reached
    imageIndex = (imageIndex + 1) % images.length;
}
```



练习15-5：创建按顺序切换的多个图像动画实例。让它们在不同的时间开始，但是它们是同步的。

提示：使用面向对象编程将图片顺序放置在一个类中。

15.5 像素，像素，以及更多的像素

如果你是按这本书的顺序认真阅读这本书，你会注意到目前位置，我们只提到了通过调用函数来进行绘制的手段。”在这些点之间绘制一条线”或者”将这个圆填充为红色”又或者”加载JPG图像并且显示出来”。但是，在某个地点，某些人不得不写一些代码来转译这些调用函数，将屏幕上的像素转换成我们所要求的形状。当我们说line()的时候线不会凭空出现，它显示是因为我们将2点之间的所有像素连成了一条路径。幸运的是，我们不需要管理这些低等级的像素点。我们要感谢我们的Processing(以及Java)开发者们，能够直接利用这些绘制函数。

然而，有时候我们也会想要挣脱我们形状绘制的舒服，直接来对像素进行直接的处理。Processing通过像素数组提供了这个功能。

我们熟悉的是在二维世界中每个像素都有自己的x和y坐标。但是，数组像素只有一维，那就是线性顺序中所存储的颜色值。如图15.5所示。

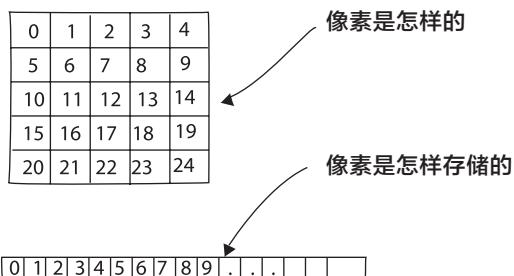


图 15.5

来看看下面的例子。在窗口中设置了每一个像素让它们都有一个随机的灰度值。像素数组就像其他数组一样，但是唯一的不同就是我们不需要声明，因为它是Processing中的内置变量。

例 15-5：设置像素

```
size(200,200);
// Before we deal with pixels
loadPixels();
// Loop through every pixel
for (int i = 0; i < pixels.length; i++) {
    // Pick a random number, 0 to 255
    float rand = random(255);
```

和其他数组一样。我们也可以得到像素数组的长度。

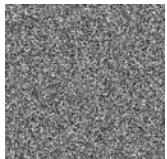


图 15.6

```

// Create a grayscale color based on random number
color c = color(rand);
// Set pixel at that location to random color
pixels[i] = c;
}

// When we are finished dealing with pixels
updatePixels();

```

通过索引，我们在像素数组中可以单独访问每一个像素元素，就和其他数组一样。

首先，我们需要指出上面例子中一些重要的东西。每当你访问像素的时候，你必须提醒Processing这个动作。通过这两个函数可以实现：

- **loadPixels()**—这个函数需要你在访问像素数组之前被调用出来，它就等于告诉Processing“加载像素，我想和它们聊聊”。
- **updatePixels()**—这个函数需要你在访问完像素数组之后被调用出来，它就等于告诉Processing“ok，更新这些像素吧，我已经改完了”。

在例15-5中，因为颜色是随机设置的，所以我们没必要关心我们所访问的像素它们的坐标。但是在许多图像处理应用中，XY坐标位置是非常重要的信息。举一个简单的例子，将每个偶数列像素点设置为白色，每个奇数列像素点设置为黑色。你怎样在一维像素数组中做到这些？你怎样知道像素的行或列？

当我们在进行像素编程时，我们需要思考在二维世界中每一个像素点，而不是直接访问一维世界中的数据。我们可以通过下面的公式来思考：

1. 假设一个窗口或一个图片，给定它们宽度和高度。
2. 然后我们就可以知道像素数组合计的元素等于宽*高。
3. 对于任何窗口中给定的X,Y点，在我们一维像素数组中就是：**LOCATION=X+Y * WIDTH**

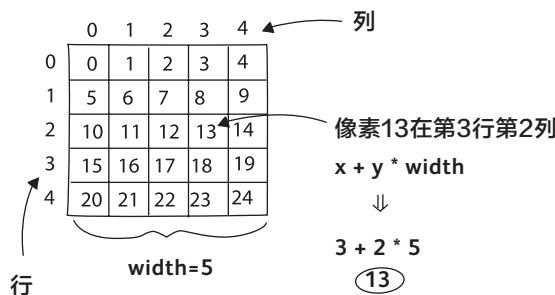


图 15.7

这可能是让你想起了第13章中的二维世界数组。实际上，我们需要使用相同的for循环技术。不同的是，虽然我们想使用for循环，但是它们是生活在一个一维世界中，所以我们需要陶俑公式，如图15.7所示。

让我们来看看它是如何完成奇数列和偶数列的问题的。如图15.8所示。

例 15-6：利用2D中的坐标设置像素

```

size(200,200);
loadPixels();

// Loop through every pixel column
for (int x = 0; x < width; x++) {
    // Loop through every pixel row
    for (int y = 0; y < height; y++) {
        // Use the formula to find the 1D location
        int loc = x + y * width;

        if (x % 2 == 0) { // If we are an even column
            pixels[loc] = color(255);
        } else { // If we are an odd column
            pixels[loc] = color(0);
        }
    }
}

updatePixels();

```

2个循环能够让我们访问每一列(x)以及每一行(y)。

通过我们的公式计算出像素的一维位置：**1D位置=x+y*width**

我们使用列数(x)来确定那些像素来填充黑色或白色。

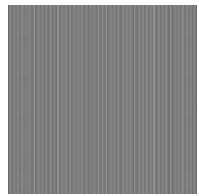
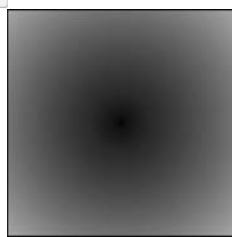


图 15.7

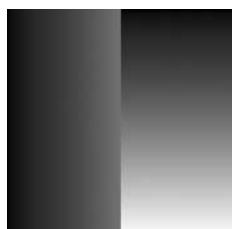
练习15-6：对应下面图片所示，完成下面的填空。



```

size(255,255);
_____;
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        int loc = _____;
        float distance = _____;
        pixels[loc] = _____;
    }
}
_____;

```



```

size(255,255);
_____;
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        _____;
        if (_____) {
            _____;
        } else {
            _____;
        }
    }
}
_____;

```

15.6 Processing 图像处理介绍

在之前的章节例子中我们学习了如果通过公式的计算来设置像素。接下来我们将学习如果通过这些存在的 PImage 对象来设置像素。下面是一些伪代码。

1. 向 PImage 对象中加载图像文件。
2. 对于 PImage 中的每一个像素，我们需要检索像素的颜色设置这些颜色。
3. PImage 的类包含了一些非常有用的内容，它能存储图像相关的信息—宽度、高度以及像素。正如我们自定义的类，我们可以通过点语法访问这些内容。

```
PImage img = createImage(320,240,RGB); // Make a PImage object
println(img.width); // Yields 320
println(img.height); // Yields 240
img.pixels[0] = color(255,0,0); // Sets the first pixel of the image to red
```

访问这些内容能够让我们检索图片所有的像素，并且将它们显示在屏幕上。

例 15-7：在图像中显示像素

```
PImage img;
void setup() {
  size(200,200);
  img = loadImage("sunflower.jpg");
}

void draw() {

  loadPixels();
  // Since we are going to access the image's pixels too
  img.loadPixels();
  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      int loc = x + y*width;

      // Image Processing Algorithm would go here
      float r = red(img.pixels[loc]);
      float g = green(img.pixels[loc]);
      float b = blue (img.pixels[loc]);

      // Image Processing would go here

      // Set the display pixel to the image pixel
      pixels[loc] = color(r,g,b);
    }
  }
  updatePixels();
}
```



图 15.9

我们同样必须通过 `loadPixels()` 函数来读取 PImage 中的像素。

函数 `red()`, `green()`, `blue()` 是从像素中检索到的颜色值。

如果我们想改变 RGB 的值，我们就要从显示这些像素之前开始下手。

现在，我们当然简化这些内容直接显示图像(例如，嵌套循环是不需要的，`image()`函数能够让我们直接跳过这些像素进行工作)但是，在例15-7中，为我们提供了通过每个像素获取它们的红，绿，蓝值的基本框架。最终，这样能够让我们开发更多更高级的图像算法框架。

在我们继续学习之前，我要强调的是这个例子，因为它显示的尺寸和图片本身的尺寸相同。如果不相同，你可能需要2个像素计算，一个是源文件，一个是显示尺寸。

```
int imageLoc = x + y * img.width;
int displayLoc = x + y * width;
```



练习15-7：使用例15-7，在显示图像之前，改变里面RGB的值。

15.7 开发属于我们自己的 `tint()` 图像滤镜工具

在之前的章节例子中我们学习了如果通过公式的计算来设置像素。接下来我们将学习如果通过这些存在的`PIImage`对象来设置像素。下面是一些伪代码。

1. 向`PIImage`对象中加载图像文件。
2. 对于`PIImage`中的每一个像素，我们需要检索像素的颜色设置这些颜色。

`PIImage`的类包含了一些非常有用的内容，它能存储图像相关的数据—宽度、高度以及像素。正如我们自定义的类，我们可以通过点语法访问这些内容。

例 15-8：调整图片的亮度

```
for (int x = 0; x < img.width; x++) {
    for (int y = 0; y < img.height; y++) {
        // Calculate the 1D pixel location
        int loc = x + y * img.width;
        // Get the R,G,B values from image
        float r = red(img.pixels[loc]);
        float g = green(img.pixels[loc]);
        float b = blue(img.pixels[loc]);
        // Change brightness according to the mouse here
        float adjustBrightness = ((float) mouseX / width) * 8.0;
        r *= adjustBrightness;
        g *= adjustBrightness;
        b *= adjustBrightness;
        // Constrain RGB to between 0-255
        r = constrain(r, 0, 255);
        g = constrain(g, 0, 255);
        b = constrain(b, 0, 255);
        // Make a new color and set pixel in the window
        color c = color(r, g, b);
        pixels[loc] = c;
    }
}
```

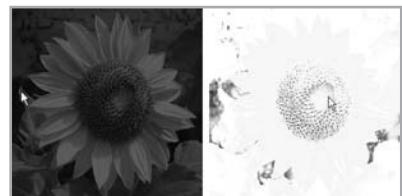


图 15.10

我们可以利用乘法，基于`mouseX`的位置计算出乘以8的值。这个结果会改变每个像素RGB的值。

RGB值被限制为0到255之间的值。

由于我们是基于每一个像素进行修改，所有的像素都不需要完全相等。例如，我们可以通过鼠标于每个像素之间的距离来修改每个像素的亮度。

例 15-9：基于像素的位置修改图片的亮度

```
for (int x = 0; x < img.width; x++) {
    for (int y = 0; y < img.height; y++) {
        // Calculate the 1D pixel location
        int loc = x + y * img.width;
        // Get the R,G,B values from image
        float r = red(img.pixels[loc]);
        float g = green(img.pixels[loc]);
        float b = blue(img.pixels[loc]);
        // Calculate an amount to change brightness
        // based on proximity to the mouse
        float distance = dist(x,y,mouseX,mouseY);
        float adjustBrightness = (50-distance)/50;
        r *= adjustBrightness;
        g *= adjustBrightness;
        b *= adjustBrightness;
        // Constrain RGB to between 0-255
        r = constrain(r,0,255);
        g = constrain(g,0,255);
        b = constrain(b,0,255);
        // Make a new color and set pixel in the window
        color c = color(r,g,b);
        pixels[loc] = c;
    }
}
```

像素离鼠标位置越近，其距离的值就会越短。如果我们希望越近的像素越亮，我们就可以用一个反转公式：

$$\text{adjustBrightness}=(50-\text{distance})/50$$
 让距离大于50(或者等于50)，像素点的亮度就等于0(负数在这里同样等于0)并且当像素点与鼠标的距离为0时，其亮度值就等于1.0。



练习 15-8：根据鼠标的交互分别修改亮度的红色，绿色以及蓝色。例如，让 `mouseX` 控制红色，`mouseY` 控制绿色，`distance` 控制蓝色，等等。

15.8 书写另一种 PImage 对象像素

我们已经学习了从原图片中用每个像素处理我们的图片，并且在窗口中直接显示修改后的像素。但是，我们通常有更方便的写法让新像素直接到达目标图片(那就是直接使用`image()`函数)。当我们进行另一个像素操作例子：阈值时我们回证明这种写法的易用性。

阈值滤镜器只会显示图片中像素的两种状态，黑色或者白色。这个状态时通过特定的阈值来进行设定。如果像素的亮度值大于阈值，那个像素点就是白色，如果小于阈值就是黑色。例15-10使用了一个任意的阈值100。



图 15.12

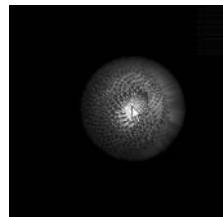


图 15.11

例 15-10：亮度阈值

```

PImage source; // Source image
PImage destination; // Destination image

void setup() {
    size(200,200);
    source = loadImage("sunflower.jpg");
    destination = createImage(source.width, source.height, RGB);
}

void draw() {
    float threshold = 127;
    // We are going to look at both image's pixels
    source.loadPixels();
    destination.loadPixels();
    for (int x = 0; x < source.width; x++) {
        for (int y = 0; y < source.height; y++) {
            int loc = x + y * source.width;
            // Test the brightness against the threshold
            if (brightness(source.pixels[loc]) > threshold){
                destination.pixels[loc] = color(255); // White
            } else {
                destination.pixels[loc] = color(0); // Black
            }
        }
    }
    // We changed the pixels in destination
    destination.updatePixels();
    // Display the destination
    image(destination,0,0);
}

```

我们需要两个图片，一个为源文件(原始文件)以及一个目标图片(用于显示)。

目标图片用来创建一个和源图片相同尺寸的空白图片。

brightness()会返回一个0到255之间的值，这是这个像素颜色的整体亮度。如果它大于100，转换成白色，如果小于100，转换成黑色。

这个写到目标图片上

我们必须显示目标图片！才能有效果



练习15-9：利用鼠标的位置控制阈值。

这个特殊的功能不能作为filter()函数的一部分处理每个像素。如果你了解底层代码你就会发现其实你完全可以制作自己的处理算法，不需要任何filter()。

如果你想做阈值，例15-11会更简化它。

例 15-11：有filter的亮度阈值

```

// Draw the image
image(img,0,0);
// Filter the window with a threshold effect
// 0.5 means threshold is 50 % brightness
filter(THRESHOLD,0.5);

```

更多关于filter():

```
filter(mode);
filter(mode,level)
```

filter()函数提供了一整套滤镜处理系统用于窗口显示。这样我们就没必要使用2个图片，滤镜系统会直接在显示的时候执行这些滤镜效果。除了 THRESHOLD 其他可用的滤镜模式包括GRAY, INVERT, POSTERIZE, BLUR, OPAQUE, ERODE, 以及 DILATE。详细内容请参考Processing reference (http://processing.org/reference/filter_.html) 中的每个具体例子。

15.9 等级二：像素组处理

在之前的例子中，我们已经看到了源图片和目标图片像素一对一的关系转变。为了提高图片的亮度，我们收集源图片像素点的RGB值，并且经过修改后，让目标图片显示出效果。为了让我们进行更高级的图像处理函数，我们必须超越一对一的像素处理，变成像素组处理。

让我们从源图片的2个像素上创建一个新的像素——一个像素和它左边的像素。

如果我们知道像素的坐标(x,y):

```
int loc = x + y*img.width;
color pix = img.pixels[loc];
```

那么它左边的那个像素就是(x-1,y):

```
int leftLoc = (x-1) + y*img.width;
color leftPix = img.pixels[leftLoc];
```

我们可以创建一个之前那个像素和它左边像素之差颜色的像素。

```
float diff = abs(brightness(pix) - brightness(leftPix));
pixels[loc] = color(diff);
```

例 15-12显示了这整个算法，结果如图15.13所示。

例 15-12：像素之差(锯齿)

```
// Since we are looking at left neighbors
// We skip the first column
for (int x = 1; x < width; x++) {
    for (int y = 0; y < height; y++) {
        // Pixel location and color
        int loc = x + y*img.width;
        color pix = img.pixels[loc];
        // Pixel to the left location and color
        int leftLoc = (x-1) + y*img.width;
        color leftPix = img.pixels[leftLoc];
```

读取像素左边的那个像素

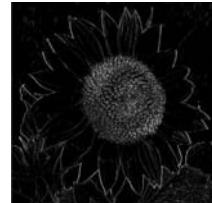


图 15.13

```
// New color is difference between pixel and left neighbor
float diff = abs(brightness(pix) - brightness(leftPix));
pixels[loc] = color(diff);
}
}
```

例15-12是一个简单的垂直锯齿检测算法。当相邻像素差距很大时，它们就会很像”锯齿”像素。例如，你想象一个白色的图片在一个黑色的左面上。它们之间的锯齿部分是最不同的，也就是黑白相交的位置。

在例15-12中，我们会在2个像素之间看到锯齿。但是有更复杂的算法，因为每个像素相邻的像素有8个：左上，上，右上，右，右下，下，左下和左。如图15.14所示。

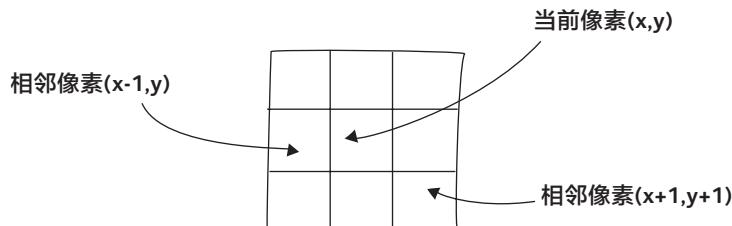


图 15.14

这些图片处理算法通常被称为”空间转换”这个过程就是在输入的像素与相邻像素之间添加一个平均值像素来进行输出。换句话说，新的像素是之前像素区域的一个函数。相邻区域之间的尺寸可以采用差值。如 3×3 矩阵 5×5 矩阵。

不同组合的平均值最后的效果也不同。例如，我们”锐化”是计算相邻像素相减的值，并且增加一个中心点像素。模糊是通过取所有像素的平均值进行计算。(需要注意的是转换矩阵值的总和为1)

例如

锐化:

-1	-1	-1
-1	9	-1
-1	-1	-1

模糊:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

例15-13利用2D数组存储一个尺寸为3*3大小的矩阵进行了一次转换(回顾2D数组请参考第13章)。这个例子可能是我们在这本中这么久遇见的最高级的例子，因为它涉及到了很多的元素(嵌套循环，2D 数组，PImage像素，等等)。

例 15-13：锐化转换

```
PImage img;
int w = 80;

// It's possible to perform a convolution
// the image with different matrices

float[][] matrix = { { -1, -1, -1 },
                     { -1, 9, -1 },
                     { -1, -1, -1 } };

void setup() {
    size(200,200);
    img = loadImage("sunflower.jpg");
}

void draw() {
    // We're only going to process a portion of the image
    // so let's set the whole image as the background first
    image(img,0,0);

    // Where is the small rectangle we will process
    int xstart = constrain(mouseX-w/2,0,img.width);
    int ystart = constrain(mouseY-w/2,0,img.height);
    int xend = constrain(mouseX + w/2,0,img.width);
    int yend = constrain(mouseY + w/2,0,img.height);
    int matrixsize = 3;
    loadPixels();
    // Begin our loop for every pixel
    for (int x = xstart; x < xend; x++) {
        for (int y = ystart; y < yend; y++) {
            color c = convolution(x,y,matrix,matrixsize,img);
            int loc = x + y * img.width;
            pixels[loc] = c;
        }
    }
    updatePixels();
    stroke(0);
    noFill();
    rect(xstart,ystart,w,w);
}

color convolution(int x, int y, float[][] matrix, int matrixsize, PImage img) {
    float rtotal = 0.0;
    float gtotal = 0.0;
    float btotal = 0.0;
    int offset = matrixsize / 2;
    // Loop through convolution matrix
    for (int i = 0; i < matrixsize; i++) {
        for (int j = 0; j < matrixsize; j++) {
            // What pixel are we testing

```

转换矩阵将储存为3*3
的”锐化”二维函数。



图 15.15

在这个例子中我们只锐化围绕鼠标坐标的80*80像素的图片部分，而不是整个图片。

每一个象素点location(x,y)
都会通过调用函数convolution()
返回一个新的颜色值用于显示。

```

int xloc = x + i-offset;
int yloc = y + j-offset;
int loc = xloc + img.width*yloc;
// Make sure we haven't walked off the edge of the pixel array
loc = constrain(loc,0,img.pixels.length-1);
// Calculate the convolution
rtotal += (red(img.pixels[loc]) * matrix[i][j]);
gtotal += (green(img.pixels[loc]) * matrix[i][j]);
btotal += (blue(img.pixels[loc]) * matrix[i][j]);
}
}

// Make sure RGB is within range
rtotal = constrain(rtotal,0,255);
gtotal = constrain(gtotal,0,255);
btotal = constrain(btotal,0,255);
// Return the resulting color
return color(rtotal,gtotal,btotal);
}

```

我们要时常注意相邻像素确保它们之间没有出现锯齿的产生。

我们计算了所有相邻的像素的总和，并且乘以转换矩阵中的值。

它们的总和被限制在0-255之间，这是一个新的颜色，并且会返回



练习 15-10：为转换矩阵尝试一下不同的值。



练习 15-11：请使用我们在例子中建立的图像处理框架，创建一个滤镜器让2个图片变成一个图片。换句话说，新图片像素的显示值应该取决于之前2个图片的像素的值。例如，你可以尝试写代码将2个图片融合在一起(不要使用 tint())。

15.10 创意可视化

你可能会说：“哇！这真的非常有趣，但是说实话，如果我们想改变图片的亮度或者让它模糊我们真的必须要写代码吗？难道不能用Photoshop？”事实上，我们现在所学习到的仅仅是了解了Adobe高级程序员的一些入门。然而，Processing真正的力量在于实时图形交互！这不需要我们一直学习象素点以及像素组的过程中。

下面有2个例子是通过算法绘制Porcessing的图形。代替了过去我们用随机填充颜色或者是用硬代码填充颜色，我们选择了PImage中像素的颜色。这个图像本身不会显示。它会变成为一个信息数据库，我们能够利用自己创造性的探索进行有趣的实验。

在第一个例子中，对于每一次draw()的循环周期，我们都会在屏幕随机的位置填充一个圆，这个圆的颜色取决于其相应位置源图像的颜色。结果就是“点彩”效果的图片。如图15.16所示。

例 15-14：点彩

```

PImage img;
int pointillize = 16;

void setup() {
    size(200,200);
    img = loadImage("sunflower.jpg");
    background(0);
    smooth();
}

void draw() {
    // Pick a random point
    int x = int(random(img.width));
    int y = int(random(img.height));
    int loc = x + y*img.width;

    // Look up the RGB color in the source image
    loadPixels();
    float r = red(img.pixels[loc]);
    float g = green(img.pixels[loc]);
    float b = blue(img.pixels[loc]);
    noStroke();

    // Draw an ellipse at that location with that color
    fill(r,g,b,100);
    ellipse(x,y,pointillize,pointillize);
}

```

回到形状！这里我们使用一个圆来代替像素。

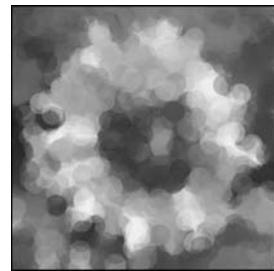


图 15.16

在接下来的例子中，我们会提取二维图片的数据，并让他3D化。让它为每一个像素渲染一个三味的立方体。Z轴的值确定其颜色亮度。越亮的点越接近我们视角，越暗的就离我们视角越远。

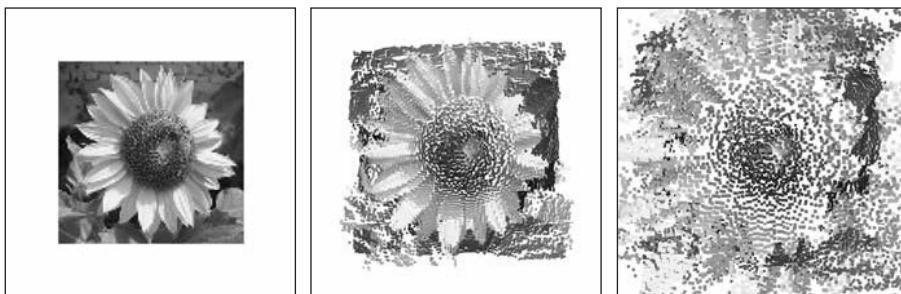


图 15.17

例 15-15：2D图像映射到3D

```

PImage img; // The source image
int cellsize = 2; // Dimensions of each cell in the grid
int cols, rows; // Number of columns and rows in our system

void setup() {
    size(200,200,P3D);
}

```

```

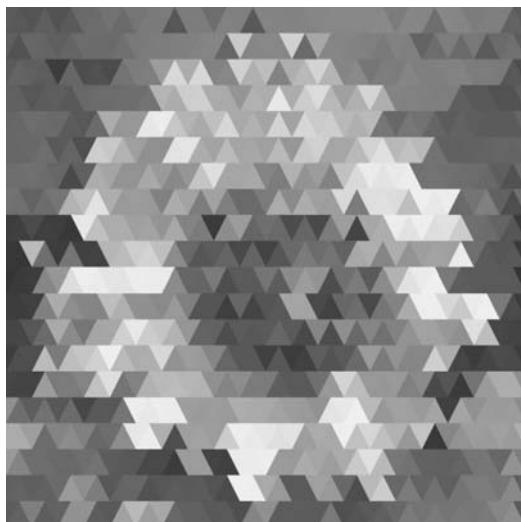
img = loadImage("sunflower.jpg"); // Load the image
cols = width/cellsize; // Calculate # of columns
rows = height/cellsize; // Calculate # of rows
}

void draw() {
    background(0);
    loadPixels();
    // Begin loop for columns
    for (int i = 0; i < cols; i++) {
        // Begin loop for rows
        for (int j = 0; j < rows; j++) {
            int x = i*cellsize + cellsize/2; // x position
            int y = j*cellsize + cellsize/2; // y position
            int loc = x + y*width; // Pixel array location
            color c = img.pixels[loc]; // Grab the color
            // Calculate a z position as a function of mouseX and pixel brightness
            float z = (mouseX/(float)width) * brightness(img.pixels[loc]) - 100.0;
            // Translate to the location, set fill and stroke, and draw the rect
            pushMatrix();
            translate(x,y,z);
            fill(c);
            noStroke();
            rectMode(CENTER);
            rect(0,0,cellsize,cellsize);
            popMatrix();
        }
    }
}

```

3D平移所计算出的z轴值做为一个关于像素亮度以及鼠标坐标的函数。

练习15-12：创建一个新的草图，利用你自定义的形状来覆盖整个窗口。加载图像并且通过图像像素的颜色填充到你自定义的形状中。例如下面利用三角形的方式。



16. 视频

“当像素出现时，它就填满了我的生活。我可能需要几十年才能消化这些在我手指和脑中的像素。”

— John Maeda

本章内容：

- 显示视频直播
- 显示录制的视频
- 创建一个软件镜像
- 将摄像头做为传感器

16.1 学习之前

现在我们已经在Processing中探索了静态图片，我们现在已经准备好学习动态图像，特别是实时摄像(以及之后，我们会从一个视频进行修改)。在我们写代码之前我们要学会使用PC或Mac的摄像头，这需要几个步骤。

在Mac上：

- 将摄像头连接到你的电脑。
- 确保安装好关于摄像头的驱动或者软件。
- 在其他程序中测试摄像头，如iChat，确保它能够正常使用。

在PC上：

- 将摄像头连接到你的电脑。
- 确保安装好关于摄像头的驱动或者软件。
- 测试摄像头(运行可以使用摄像头的软件)。
- 安装QuickTime(version 7或更高版本)，如果你使用之前的版本，你必须选择”自定义”安装，并且勾选“QuickTime for Java”。
- 你同样需要安装VDIG(视频数字转换器)，它能够让QuickTime在窗口中捕捉视频。VDIG会不断更新的，所以我建议你看看这本书的网站获取最新的版本。虽然这个软件没有继续被开发，但是你能下到免费的VDIG:<http://eden.net.nz/7/20071008/>。你同样也能考虑使用Abstract Plane's vdig (<http://www.abstractplane.com.au/products/vdig.jsp>)，它需要支付一小笔费用，但是具有很多高级功能。确保安装好关于摄像头的驱动或者软件。

不可否认，在Mac上的过程比较简单。这是因为Processing使用了Java中的QuickTime库处理视频。在苹果电脑上，QuickTime是预装软件(虽然有时更新会导致出现问题)，而在Windows上，我们需要按照安装步骤来正确安装和配置QuickTime。在本章结尾，我们会提供一些Processing中的第三方库，它们在Windows的情况下是不需要QuickTime来捕捉视频的。

 练习16-1：让摄像头连接到你的电脑。看看它能不能在其他程序上运行？如果是在PC上，请安装DVIG，然后再测试。

16.1 实时视频 101

一旦你确保了摄像头能够连接到你的电脑并正常工作，你就能开始编写Processing代码来捕捉和显示图像了。我们开始通过基本步骤导入视频库以及使用Capture类来显示实时视频

第一步：导入Processing的Video库

如果你跳过了第12章Processing库的学习，你可能需要回去学习这个章节。虽然没有太多内容，但是一旦video库来到，你就需要导入相关的库。这里可以选择“menu”选项Sketch→Import Library→Video进行导入，或者手动输入下列一行代码(这行代码应该放置于草图的顶部)：

```
import processing.video.*;
```

使用“Import Library”菜单除了插入那一行代码以外不会有其他能做的，所以通常来说用手打代码更好。

第二步：声明Capture对象

我们在第12章中学习了如果从Processing中的内置类中创建对象。例如，我们从Processing的PIImage类中创建PIImage对象。值得注意的是，这个类是processing.core库中的一部分，并且这个库不需要任何导入，它是默认导入的。video库中包含2个非常有用的类，第一个是Capture，主要针对实时视频；第二个是Movie，主要针对录制视频。我们将开始声明一个Capture对象。

```
Capture video;
```

第三步：初始化Capture对象

Capture对象“video”就和其他的对象一样。正如我们在第8章中学习的一个对象的构造，我们会使用new运算符。如果你需要在草图开始的时候捕捉视频，Capture对象的代码通常就会出现在setup()中。

```
video = new Capture();
```

上面的那行代码缺少构造参数。请记住，这不是一个我们自己写的类，所以我们需要通过查看reference知道括号中需要怎样的参数。关于Capture函数构造器的在线参考可以访问Processing网站：

<http://www.processing.org/reference/libraries/video/Capture.html>

reference将告诉我们关于调用函数构造器的几种方法(关于超载多个函数构造器请参考第23章)。通常的方法是通过调用Capture函数构造器添加4个参数。

```
void setup() {
    video = new Capture(this,320,240,30);
}
```

让我们来看看Capture函数构造器中的参数使用。

- **this**—如果你对这个词感到困惑并不要紧张。因为到目前为止书中的例子还没有提到过这个词。从技术上讲，它指的是一个类的实例。不幸的是这种定义方式很可能会让你头疼。一种更好的方式是自我引用语句。毕竟，如果你需要在你Processing编程语句中给一个简称，你一般都会说” me” 或” I”，当然这些话在Java中是不存在的，所以我们需要用” this” 来代替。因为我们需要通过” this” 告诉Capture对象：“ 听着，我想要捕捉视频，并且在摄像头中出现新图像时提醒我。”

- **320**—幸运的是第一个参数是this，也只是唯一让人头疼的参数。320表示我们捕捉视频的宽度。

- **240**—捕捉视频的高度。

- **30**—所期望捕捉视频的帧率，每秒帧数(fps)。实际上你需要的帧率取决于你需要干什么。如果你只是在每隔一段时间捕捉一些图片，那么帧率不需要那么高。但是如果你希望以流畅的动态显示出来，那么30是一个不错的选择。当然，这仅仅是你的期望的帧率。如果你的电脑很慢，或者你需要的视频质量太高，帧率就可能达不到。

第四步：从摄像头读取图片

这里有两种方法能够让我们从摄像头读取帧。我们会简要地看看这两种方法，并且选择其中一个做为本章的例子。虽然是两种方法，但是它们都是根据相同的基本原则来读取：当一个新的帧可以被读取时，我们只会从摄像头中读取一张图片。

为了检查图片是否可用，我们需要使用available()函数，它会返回true或者false，这取决于它是否存在。如果存在，那么read()函数被调用，并且帧会从摄像头的内存中被读取。我们需要在draw()循环中不断重复这个动作，检查是否有新图像存在，用于我们读取。

```
void draw() {
    if (video.available()) {
        video.read();
    }
}
```

第二种方法，通过”event”，需要一个函数执行目标事件，在这种情况下，就是摄像头事件。如果你还记得第3章函数mousePressed()是当鼠标按下时被执行。同样，对于视频来说，我们可以选择函数captureEvent()，它能够在任何时候引用捕捉事件，一个来自摄像头的帧。这就是事件函数(mousePressed(),keyPressed(),captureEvent(),等等)有时候我们也称之为”回调”。捕捉对象”video”知道通知小程序来引用captureEvent()，因为我们通过参考自身来创建”video”。

captureEvent()是一个函数，所以它需要有自己的代码块，并独立于setup()和draw()之外。

```
void captureEvent(Capture video) {
    video.read();
}
```

总而言之，每当我们想调用函数read()时，我们需要通过使用available()来进行检查，或者是做一个回调处理—captureEvent()。还有一些其他的库我们将在之后的章节进行探索(如网络和串)它们的工作方式时完全相同的。

第五步：显示视频图像

毫无疑问，这是最简单的部分。我们可以将Capture对象认为是随时间变化的PImage，实际上Capture对象能够利用相同的方式做为PImage对象使用。

```
image(video,0,0);
```

这些所有的步骤放在一起就形成了例16-1

例 16-1：显示视频

```
// Step 1. Import the video library
import processing.video.*;  
  
// Step 2. Declare a Capture object
Capture video;  
  
void setup() {
    size(320,240);
    // Step 3. Initialize Capture object via Constructor
    // video is 320 x 240, @15 fps
    video = new Capture(this,320,240,15);
}
```

第一步，导入 video 库

第二步，声明一个 Capture 对象

第三步，初始化 Capture 对象！这样捕捉视频的过程就开始了。



图 16.1

```

void draw() {
    // Check to see if a new frame is available
    if (video.available()) {
        // If so, read it.
        video.read();
    }
}

// Display the video image
image(video,0,0);
}

```

第四步，从摄像头读取图像。

第五步，显示图像。

同样我们也可以在捕捉视频的时候做一个PImage（调整大小，移动等等）。只要我们从read()中读取图片，视频就会随着我们的鼠标进行改变。如例16-2所示。

例 16-2：操作视频图像

```

// Step 1. Import the video library
import processing.video.*;

Capture video;

void setup() {
    size(320,240);
    video = new Capture(this,320,240,15);
}

void draw() {
    if (video.available()) {
        video.read();
    }

    // Tinting using mouse location
    tint(mouseX,mouseY,255);
    // Width and height according to mouse
    image(video,0,0,mouseX,mouseY);
}

```



图 16.2

视频图片只要带有 PImage, 就可以调整它的尺寸，色调等

我们可以把视频中的每一张图片按照第15章的调整来进行编辑视频。下面是“调节亮度”的例子。

例 16-3：调整视频亮度

```

// Step 1. Import the video library
import processing.video.*;

Capture video;

void setup() {
    size(320,240);
    // Step 3. Initialize Capture object via Constructor
    video = new Capture(this,320,240,15); // video is 320 x 240, @15 fps
    background(0);
}

```

```

void draw() {
    // Check to see if a new frame is available
    if (video.available()) {
        // If so, read it.
        video.read();
    }

    loadPixels();
    video.loadPixels();
    for (int x = 0; x < video.width; x++) {
        for (int y = 0; y < video.height; y++) {
            // calculate the 1D location from a 2D grid
            int loc = x + y * video.width;
            // get the R,G,B values from image
            float r,g,b;
            r = red (video.pixels[loc]);
            g = green (video.pixels[loc]);
            b = blue (video.pixels[loc]);
            // calculate an amount to change brightness based on proximity to the mouse
            float maxdist = 100;// dist(0,0,width,height);
            float d = dist(x,y,mouseX,mouseY);
            float adjustbrightness = (maxdist-d)/maxdist;
            r *= adjustbrightness;
            g *= adjustbrightness;
            b *= adjustbrightness;
            // constrain RGB to make sure they are within 0-255 color range
            r = constrain(r,0,255);
            g = constrain(g,0,255);
            b = constrain(b,0,255);
            // make a new color and set pixel in the window
            color c = color(r,g,b);
            pixels[loc] = c;
        }
    }
}

updatePixels();
}

```



图 16.3

练习16-12：重现例15-14（点彩）来应用到实时视频中。



16.3 录制过的视频

显示已经录制过的(现成的)视频的结构大部分和实时视频内容一样。Processing video库只接受QuickTime允许的格式的视频。如果你的视频文件不是QuickTime所能允许的格式，你可能需要转换格式或者使用其他第三方库。请注意，播放现成的视频不需要VIDIG。

第一步 声明一个Movie对象

```
Movie movie;
```

第二步 初始化Movie对象

```
movie = new Movie(this, "testmovie.mov");
```

它所需要的唯一参数就是视频的文件名并且要用引号扩起来。视频文件应该保存在草图的数据目录中。

第三步 开始播放视频

在这里，我们有2种方式，play()会播放一次视频，loop()会循环播放视频。

```
movie.loop();
```

第四步 从视频中读取帧

同样，这就等同于捕捉。我们可以检查是否有新的帧出现，或者使用一个回调函数。

```
void draw() {
    if (movie.available()) {
        movie.read();
    }
}
```

或者

```
void movieEvent(Movie movie) {
    movie.read();
}
```

第五步 显示视频

```
image(movie,0,0);
```

把上述步骤放在一起如例 16-4所示。

例 16-4：显示QuickTime视频

```

import processing.video.*;

Movie movie; // Step 1. Declare Movie object

void setup() {
    size(200,200);
    // Step 2. Initialize Movie object
    movie = new Movie(this, "testmovie.mov"); // Movie file should be in data folder
    // Step 3. Start movie playing
    movie.loop();
}

// Step 4. Read new frames from movie
void movieEvent(Movie movie) {
    movie.read();
}

void draw() {
    // Step 5. Display movie.
    image(movie,0,0);
}

```

虽然Processing不能处理复杂的显示以及操作(如果有大型视频文件，显示起来也有些缓慢)，但是这里有一些video库中高级的功能可以使用。这些函数能够获取视频的时间(以秒为单位)，能够让他加速或者减速播放，也能让它直接转跳到特定的时间点播放。

下面是使用jump()(转跳特点时间点)和duration()(返回视频时间长度)的例子。

例 16-5：向前向后拖动视频

```

// If mouseX is 0, go to beginning
// If mouseX is width, go to end
// And everything else scrub in between
import processing.video.*;
Movie movie;

void setup() {
    size(200,200);
    background(0);
    movie = new Movie(this, "testmovie.mov");
}

void draw() {
    // Ratio of mouse X over width
    float ratio = mouseX / (float) width;
    // Jump to place in movie based on duration
    movie.jump(ratio*movie.duration());
    movie.read(); // read frame
    image(movie,0,0); // display frame
}

```

jump()函数允许你立即跳转到你所指定的时间点。**duration()**会返回视频的时间，单位是秒。



练习16-3：在 Movie 类中使用 speed() 函数，让程序可以根据鼠标控制视频的播放速度。注意 speed() 只需要一个函数，那个参数会乘以电影的播放速率。如果是0.5，那么视频的播放速率会慢一半；如果是2，视频的播放速度会是一倍；如果乘以-2，那么会反向播放。。

16.4 模拟软件

随着小型摄像头越来越多的普及到家用个人电脑上，开发实时的模拟软件也变的越来越流行。这些类型的应用被称之为“模拟软件”，他们提供了观看者数字图像的反馈。Processing的extensive 库也能够针对这些图形，对摄像头进行实时的捕捉，这样能够很好的为反射镜的软件设计提供一个基础。

正如我们在本章前面所看到的，我们可以应用基本的图片处理技术来处理视频图片，读取并且更换像素。将这个方法在深入一步，我们可以读取像素，并且应用颜色将他们绘制在屏幕上。

我们将从一个捕捉80*60的视频例子来讲，将它渲染在640*480的窗口中。对于视频中的每一个像素，我们讲绘制8*8的方形。

首先让我们编写显示8*8方形的网格。如图16.4所示。

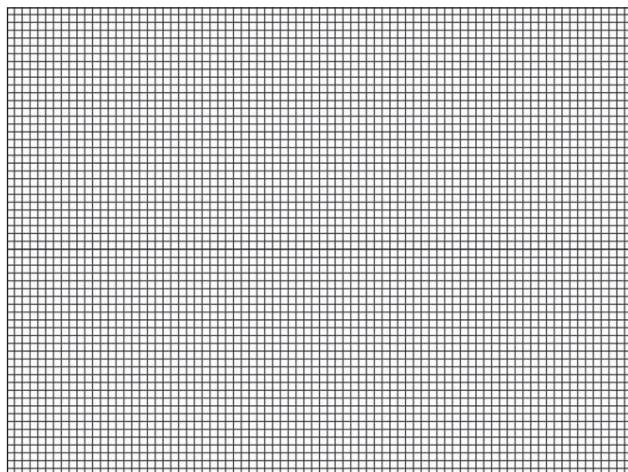


图 16.4

例 16-6：绘制8*8的方形

```
// Size of each cell in the grid, ratio of window size to video size
int videoScale = 8;
// Number of columns and rows in our system
int cols, rows;

void setup() {
    size(640,480);
    // Initialize columns and rows
    cols = width/videoScale;
    rows = height/videoScale;
}

void draw() {
    // Begin loop for columns
    for (int i = 0; i < cols; i++) {
        // Begin loop for rows
        for (int j = 0; j < rows; j++) {
            // Scaling up to draw a rectangle at (x,y)
            int x = i*videoScale;
            int y = j*videoScale;
            fill(255);
            stroke(0);
            rect(x,y,videoScale,videoScale);
        }
    }
}
```

videoScale变量告诉我们窗口尺寸与网格尺寸的比例。

对于每一行和每一列，方形都会通过**videoScale**进行缩放绘制在(x,y)位置上。

我们知道我们想有一个8*8的方形，我们可以利用列数除以8,行数也除以8。

- $640/8=80$ columns
- $480/8=60$ rows

现在我们可以捕捉的视频图像是80*60。这非常有用，因相比于捕捉640*480的视频，80*60的速度会更快。我们只需要捕捉到为我们草图所需要的色彩信息。

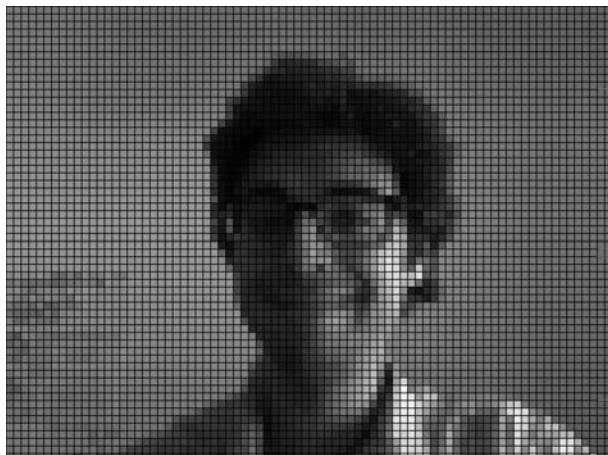


图 16.5

对于每一个 i 行， j 列的方形，我们都有其相应的像素 (i,j) 颜色与之对应。如例16-7所示(粗体为新的部分)。

例 16-7：视频像素化

```
// Size of each cell in the grid, ratio of window size to video size
int videoScale = 8;
// Number of columns and rows in our system
int cols, rows;
// Variable to hold onto Capture object
Capture video;
void setup() {
    size(640,480);
    // Initialize columns and rows
    cols = width/videoScale;
    rows = height/videoScale;
    background(0);
    video = new Capture(this,cols,rows,30);
}
void draw() {
    // Read image from the camera
    if (video.available()) {
        video.read();
    }
    video.loadPixels();
    // Begin loop for columns

    for (int i = 0; i < cols; i++) {
        // Begin loop for rows
        for (int j = 0; j < rows; j++) {
            // Where are we, pixel-wise?
            int x = i*videoScale;
            int y = j*videoScale;

            // Looking up the appropriate color in the pixel array
            color c = video.pixels[i + j*video.width];
            fill(c);
            stroke(0);
            rect(x,y,videoScale,videoScale);
        }
    }
}
```

每个方形的颜色都来自于捕捉对象的
像素数组中的颜色

正如你所看到的，只需要补充一点点内容，就能让你的网格系统填充视频的颜色。我们需要做的就是声明并且初始化Capture对象，并且读取它，让像素数组的颜色填充进去。

用较少的面积对应像素的颜色同样也是可行的。在下面的例子中，只运用了黑色和白色。视频中越亮的地方方形就越大，越暗的地方方形就越小。如图16.6所示。

例 16-6：绘制8*8的方形

```
// Each pixel from the video source is drawn as a
// rectangle with size based on brightness.
import processing.video.*;

// Size of each cell in the grid
int videoScale = 10;

// Number of columns and rows in our system
int cols, rows;

// Variable for capture device
Capture video;

void setup() {
    size(640,480);
    // Initialize columns and rows
    cols = width/videoScale;
    rows = height/videoScale;
    smooth();
    // Construct the Capture object
    video = new Capture(this,cols,rows,15);
}

void draw() {
    if (video.available()) {
        video.read();
    }

    background(0);
    video.loadPixels();
    // Begin loop for columns
    for (int i = 0; i < cols; i++) {
        // Begin loop for rows
        for (int j = 0; j < rows; j++) {
            // Where are we, pixel-wise?
            int x = i*videoScale;
            int y = j*videoScale;
            // Reversing x to mirror the image
            int loc = (video.width - i - 1) + j*video.width;
            // Each rect is colored white with a size determined by brightness
            color c = video.pixels[loc];
            float sz = (brightness(c)/255.0)*videoScale;
            rectMode(CENTER);
            fill(255);
            noStroke();
            rect(x + videoScale/2,y + videoScale/2,sz,sz);
        }
    }
}
```

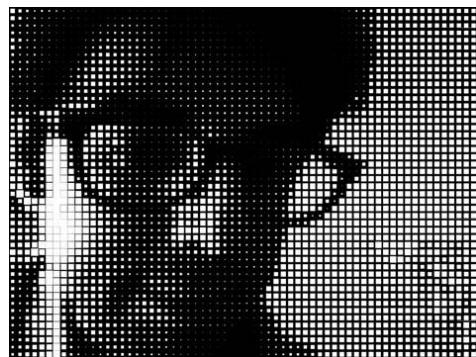


图 16.6

为了模拟图片，列必须反转公式如下： $\text{模拟列数} = \text{宽度} - \text{列数} - 1$

方形的尺寸被作为像素亮度的函数进行计算。越亮的像素显示的方形越大，越暗的像素显示的方形就越小。



练习16-4：使用例 16-9和16-10 的方法创建属于你自己的模拟软件。先创建你的图案系统，然后再使用视频以确定颜色，行为，等等。

通常开发模拟软件有两个很有用的步骤。它能够帮助你将像素匹配到网格中。

第一步，开发一个有趣的图案覆盖整个窗口

第二步，使用视频像素的颜色作为一个表格，来为这些图案上色

比方说第一步，我们随意编写一个随即的直线涂鸦。这里是我们的算法，我们用伪代码写出来。

- 以屏幕中心为x,y起始点。
- 不停重复下列动作：
 - 选择一个新的x和y(保持在屏幕内)
 - 绘制一条线从老的(x,y)到新的(x,y)
 - 保存新的(x,y)

例 16-9：随意涂鸦

```
// Two global variables
float x;
float y;

void setup() {
  size(320,240);
  smooth();
  background(255);
  // Start x and y in the center
  x = width/2;
  y = height/2;
}

void draw() {
  // Pick a new x and y
  float newx = constrain(x + random(-20,20),0,width);
  float newy = constrain(y + random(-20,20),0,height);
  // Draw a line from x,y to the newx,newy
  stroke(0);
  strokeWeight(4);
  line(x,y,newx,newy);
  // Save newx, newy in x,y
  x = newx;
  y = newy;
}
```

为了能够重复这个过程，我们在最后
存储了新的(x,y)坐标。



图 16.7

新x,y坐标是当前(x,y)坐标加或减的随机数。新的坐标被限制在窗口像素中。

现在我们已经完成了我们需要的基本图案，我们能够通过视频图片的颜色来设置stroke()的颜色。注意新添加的代码用粗体表示，如例16-10所示。

例 16-10：涂鸦模拟

```
import processing.video.*;

// Two global variables
float x;
float y;

// Variable to hold onto Capture object
Capture video;

void setup() {
    size(320,240);
    smooth();
    // framerate(30);
    background(0);
    // Start x and y in the center
    x = width/2;
    y = height/2;
    // Start the capture process
    video = new Capture(this,width,height,15);
}

void draw() {
    // Read image from the camera
    if (video.available()) {
        video.read();
    }

    video.loadPixels();
    // Pick a new x and y
    float newx = constrain(x + random(-20,20),0,width-1);
    float newy = constrain(y + random(-20,20),0,height-1);

    // Find the midpoint of the line
    int midx = int((newx + x) / 2);
    int midy = int((newy + y) / 2);

    // Pick the color from the video, reversing x
    color c = video.pixels[(width-1-midx) + midy*video.width];
    // Draw a line from x,y to the newx,newy
    stroke(c);
    strokeWeight(4);
    line(x,y,newx,newy);
    // Save newx, newy in x,y
    x = newx;
    y = newy;
}
```

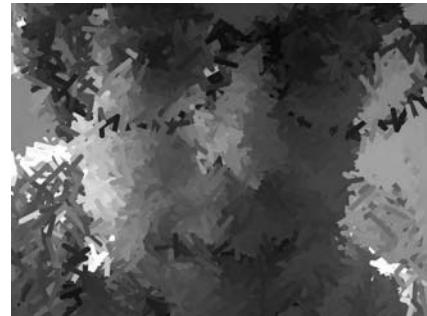


图 16.8

如果窗口很大(比如说800*600)
，我们可能会引入videoScale变量，这样能够让我们捕捉大图像。

涂鸦的颜色来自
于视频像素的颜
色



练习16-4：使用例 16-9 和 16-10 的方法，创建属于你自己的模拟软件。先创底部基本图案，然后在加上视频。

16.5 视频传感器，计算机视觉

在这一章节中，每一段视频都会被作为数据来源显示在屏幕上。这一章节我们会为你介绍不显示图片的视频，它就是“计算机视觉”，计算机视觉是一个专门致力于计算机领域的科学研究，我们会使用摄像头作为传感器。

为了更好的了解计算机视觉算法的内部运作。我们将从一个像素开始写我们的代码。但是为了进一步探索更深层次的内容，你可能需要下载一些第三方计算机视觉库。这些库都有的内容都会超出本章所学习的范围。在本章结尾，我们会简要概述这些库。

让我们从一个简单的例子开始！

摄像头使我们最好的朋友，因为它听过了大量的信息。一个320*240的图片中有76,800个像素！假如我们把所有的像素归结为一个号码：它是否就能覆盖了整个房间的亮度？这样我们就完成了一个价值1美元的光传感器(或者叫“光感元件”)，但是作为练习，我们需要让我们的摄像头完成这个功能。

我们已经在其他例子中看到我们可以使用brightness()函数检测像素的亮度，它会返回一个0到255之间的浮点数。下面几行代码可以检测视频图像中第一个像素的亮度。

```
float brightness = brightness(video.pixels[0]);
```

我们可以加入所有像素的亮度值除以所有的像素总和，就可以得出亮度的整体(即平均)值。

```
video.loadPixels();
// Start with a total of 0
float totalBrightness = 0;
// Sum the brightness of each pixel
for (int i = 0; i < video.pixels.length; i++) {
    color c = video.pixels[i];
    totalBrightness += brightness(c);
}

// Compute the average
float averageBrightness = totalBrightness / video.pixels.length;
// Display the background as average brightness
background(averageBrightness);
```

所有亮度的总和
 亮度的整体值=亮度的总和
 除以像素的总和

当你开始为这个欢呼神奇之前，你需要找一个视频来提供这些数据分析这种算法。它还没有真正发挥摄像头“看”的强大力量。毕竟，视频图像不是一个颜色拾取器，但是它是一个颜色空间趋向的集合。通过开发这种搜索像素以及识别，我们能够开发更多高级的计算机视觉应用。

跟踪最亮的颜色是一个好的第一步。想想一下在一个河南的房间里，一个移动的光源。利用这个技术，我们可以利用这种光源代替鼠标作为一种交互动作。是的，你在用你自己的手电筒玩做交互。

首先，我们讲探讨如何搜索一个图像以及找到最亮像素的x,y坐标。我们要通过一种方式来循环所有像素，找到“世界之最”最亮的那个像素(利用brightness()函数)。在最开始时，最亮的记录在第一个像素。然而不断循环，最亮的记录会被其他像素打破。在循环的结尾，记录中最亮的像素会作为图像中最亮的像素。

代码如下：

```
// The record is 0 when we first start
float worldRecord = 0.0;
// Which pixel will win the prize?
int xRecordHolder = 0;
int yRecordHolder = 0;

for (int x = 0; x < video.width; x++) {
    for (int y = 0; y < video.height; y++) {
        // What is current brightness
        int loc = x + y*video.width;
        float currentBrightness = brightness(video.pixels[loc]);
        if (currentBrightness > worldRecord) {
            // Set a new record
            worldRecord = currentBrightness;
            // This pixel holds the record!
            xRecordHolder = x;
            yRecordHolder = y;
        }
    }
}
```

这个例子中的目标，会追踪那个特定的颜色，在实际中，我们不仅仅可以跟踪最亮的。例如，我们我们可以找到最“红”或者是最“蓝”的像素。为了执行这种类型的分析，我们需要开发一种方法进行颜色比较。让我们创建颜色c1和c2。

```
color c1 = color(255,100,50);
color c2 = color(150,255,0);
```

颜色只能比较他们红色，绿色，蓝色的分量，所以首先我们必须分离出这些值。

```
float r1 = red(c1);
float g1 = green(c1);
float b1 = blue(c1);
float r2 = red(c2);
float g2 = green(c2);
float b2 = blue(c2);
```

现在，我们已经准备好比较颜色。一种方式是比较他们绝对值的总和。这很纠结，但是这个方法却很实用。r1减去r2，因为我们只关心幅度的差异，所以不用管它是正数还是负数，只需要绝对值即可。像同样的方式比较绿色和蓝色，并且将他们相加。

```
float diff = abs(r1-r2) + abs(g1-g2) + abs(b1-b2);
```

虽然这个方法不错(计算出来也很快)，但是有一种更准确的方法来计算颜色之间的“差距”。你可能在想：“更准确？颜色之间的差距应该怎样判断才好？”我们通过Pythagorean Theorem知道了2点之间距离最短。我们同样可以将颜色看作一个三围空间，让x,y,z取代r,g,b。如果两个颜色离的很近，那么他们之间的色彩差距就越小，如果越远，他们的色彩差距就越大。

```
float diff = dist(r1,g1,b1,r2,g2,b2);
```

例如寻找最“红”的像素，我们就要寻找最接近红色(255,0,0)的像素。

虽然计算更准确，但是犹豫在dist()函数中涉及到平方根的计算，这样要比绝对值算法要慢很多。解决这个问题的好方法就是将开根去掉。 $colorDistance=(r1-r1)*(r1-r1)+(g1-g1)*(g1-g1)+(b1-b1)*(b1-b1)$

通过调整亮度跟踪代码来找到跟任何给定颜色(不是最亮颜色)最接近的颜色，我们可以将颜色跟踪放草图中。下列的例子，用户可以点击图像中的一个颜色来进行跟踪。一个黑色的源泉会出现在最接近它给定的颜色的位置。如图16.9所示。

例 16-11：简单的颜色跟踪

```
import processing.video.*;
// Variable for capture device

Capture video;
color trackColor;

void setup() {
    size(320,240);
    video = new Capture(this,width,height,15);
    // Start off tracking for red
    trackColor = color(255,0,0);
    smooth();
}
```

声明一个我们在寻找的
颜色变量



图 16.9

```

void draw() {
    // Capture and display the video
    if (video.available()) {
        video.read();
    }

    video.loadPixels();
    image(video,0,0);
    // Closest record, we start with a high number
    float worldRecord = 500;
    // XY coordinate of closest color
    int closestX = 0;
    int closestY = 0;

    // Begin loop to walk through every pixel
    for (int x = 0; x < video.width; x++) {
        for (int y = 0; y < video.height; y++) {
            int loc = x + y*video.width;
            // What is current color
            color currentColor = video.pixels[loc];
            float r1 = red(currentColor);
            float g1 = green(currentColor);
            float b1 = blue(currentColor);
            float r2 = red(trackColor);
            float g2 = green(trackColor);
            float b2 = blue(trackColor);
            // Using euclidean distance to compare colors
            float d = dist(r1,g1,b1,r2,g2,b2);

            // If current color is more similar to tracked color than
            // closest color, save current location and current difference
            if (d < worldRecord) {
                worldRecord = d;
                closestX = x;
                closestY = y;
            }
        }
    }

    if (worldRecord < 10) {
        // Draw a circle at the tracked pixel
        fill(trackColor);
        strokeWeight(4.0);
        stroke(0);
        ellipse(closestX,closestY,16,16);
    }
}

void mousePressed() {
    // Save color where the mouse is clicked in trackColor variable
    int loc = mouseX + mouseY*video.width;
    trackColor = video.pixels[loc];
}

```

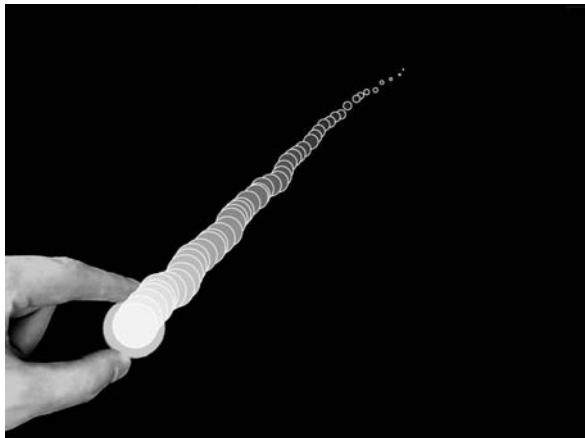
在我们开始寻找之前，现在的颜色记录应该设置为一个较高的数字，这样会比设置为第一个像素容易很多。

我们使用dist()函数来比较当前像素颜色与我们需要跟踪像素的颜色。

我们只考虑2个颜色之间的距离小于10的颜色。这个阈值10是任意的，你也可以改成其他你需要规定的范围。



练习16-5：使用任何你之前创建的草图，让我们的颜色跟踪代替鼠标交互。为摄像头创建一个简单的以及对比度高的环境。例如，将摄像头的镜头对准一个黑色背景中的白色物件。控制你草图中对象的位置。下面的图中说明用一个瓶盖了控制“蛇”(例9-9)。



16.5 消除背景

颜色差异的对照，对于计算机视觉是一种非常有用的算法。比如说你想让一段你跳草裙舞的视频背景进行转化，不是再办公室，而是在沙滩上。消除背景技术能够让你删除这些图像的背景，并且用任何你喜欢的像素去代替(如沙滩)，同时前景(你的舞蹈)完好无损。

下面是我们的算法：

- 记住背景图像
- 检查当前视频帧中的每一个像素。如果和记住的背景图像有很多的不同，那么清除这些像素，他们是背景像素。仅仅显示前景像素。

为了证明上述的算法，我们将执行一个反向的绿色屏幕。草图中我们会消除背景，并且用绿色像素代替。

第一步是“纪录”背景。背景基本上是一个时评的快照。由于视频图像会不断变化，我们必须单独的复制一帧视频作为单独的PImage对象。

```
PImage backgroundImage;
```

```
void setup() {
    backgroundImage = createImage(video.width, video.height, RGB);
}
```

当backgroundImage对象被创建时，它是一个空白的图像，它的尺寸和视频相同。这种形式不是特别有用，所以当我们想纪录背景时，我们需要从摄像头的图像中复制一个背景图像。当我们点击鼠标时，就开始实施这个行为。

```
void mousePressed() {
    // Copying the current frame of video into the backgroundImage object
    // Note copy takes 5 arguments:
    // The source image
    // x,y,width, and height of region to be copied from the source
    // x,y,width, and height of copy destination
    backgroundImage.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);
    backgroundImage.updatePixels();
}
```

copy()函数允许我们将一个图像的像素复制到另一个图像上。注意在新像素被复制之后**updatePixels()**应该被调用。

一旦我们将背景图像保存，我们就能在当前帧中循环所有像素，并且比较他们与背景图像之间的颜色差别。对于任意给定的像素(x,y)，我们使用如下代码：

```
int loc = x + y*video.width; // Step 1, what is the 1D pixel location
color fgColor = video.pixels[loc]; // Step 2, what is the foreground color
color bgColor = backgroundImage.pixels[loc];// Step 3, what is the background color

// Step 4, compare the foreground and background color
float r1 = red(fgColor); float g1 = green(fgColor); float b1 = blue(fgColor);
float r2 = red(bgColor); float g2 = green(bgColor); float b2 = blue(bgColor);
float diff = dist(r1,g1,b1,r2,g2,b2);

// Step 5, Is the foreground color different from the background color
if (diff > threshold) {
    // If so, display the foreground color
    pixels[loc] = fgColor;
} else {
    // If not, display green
    pixels[loc] = color(0,255,0);
}
```

上面的代码有一个假定变量”threshold”（阈值）。阈值越小，前景图像就越容易显示。我们不必与背景图像差异特别大。下面是将threshold作为一个全局变量的完整例子。

例 16-12：简单的背景消除

```
// Click the mouse to memorize a current background image
import processing.video.*;
// Variable for capture device
Capture video;
// Saved background
PI mage backgroundImage;
// How different must a pixel be to be a foreground pixel
float threshold = 20;
```



图 16.10

```

void setup() {
    size(320,240);
    video = new Capture(this, width, height, 30);
    // Create an empty image the same size as the video
    backgroundImage = createImage(video.width,video.height,RGB);
}

void draw() {
    // Capture video
    if (video.available()) {
        video.read();
    }
    loadPixels();
    video.loadPixels();
    backgroundImage.loadPixels();
    // Draw the video image on the background
    image(video,0,0);
    // Begin loop to walk through every pixel
    for (int x = 0; x < video.width; x++) {
        for (int y = 0; y < video.height; y++) {
            int loc = x + y*video.width; // Step 1, what is the 1D pixel location
            color fgColor = video.pixels[loc]; // Step 2, what is the foreground color
            // Step 3, what is the background color
            color bgColor = backgroundImage.pixels[loc];
            // Step 4, compare the foreground and background color
            float r1 = red(fgColor);
            float g1 = green(fgColor);
            float b1 = blue(fgColor);
            float r2 = red(bgColor);
            float g2 = green(bgColor);
            float b2 = blue(bgColor);
            float diff = dist(r1,g1,b1,r2,g2,b2);
            // Step 5, Is the foreground color different from the background color
            if (diff > threshold) {
                // If so, display the foreground color
                pixels[loc] = fgColor;
            } else {
                // If not, display green
                pixels[loc] = color(0,255,0);
            }
        }
    }
    updatePixels();
}

void mousePressed() {
    // Copying the current frame of video into the backgroundImage object
    // Note copy takes 5 arguments:
    // The source image
    // x,y,width, and height of region to be copied from the source
    // x,y,width, and height of copy destination
    backgroundImage.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);
    backgroundImage.updatePixels();
}

```

我们正在寻找视频像素，纪录
backgroundImage的像素，
并且访问这些显示的像素。因此
我们必须loadPixels()他们全
部！

我们可以选择用一些除了绿色以外的颜
色来替换背景像素。

当你最后运行这个例子，先走出视频框架，点击鼠标，纪录下没有你的时候的背景，然后回到框架中。你就回到图16.10的结果。

如果这个草图不能正常运行工作，检查看看是不是你的摄像头的“automatic(自动)”功能被开启。例如，如果你摄像头设置了自动调整亮度或者白平衡，你的草图运行就会出现问题。虽然你的背景图像被纪录了，但是旦整个图像变量或者改变了色调，草图就会问你所有的像素都已经改变，也包括了前景部分！所以为了达到最佳的效果，请停止使用您摄像头的任何自动功能。



练习16-6：将替换绿色像素改变成替换其他图片。当阈值在什么时候运行效果最好？当阈值在什么时候运行效果最不好？尝试利用鼠标控制阈值变量。

16.7 运动检测

今天是高兴的一天。为什么？因为我们做了所有的工作去了解如何从视频中删除背景，让我们检测自由运动。在背景消除的例子中，我们研究了每个像素的关系并储存了一个背景图片。当像素颜色有很大改变时，运动在视频中就会经常产生。换句话说，运动检测是和消除背景完全一样的算法，只是代替了保存背景图像，我们保存会不断保存之前1帧的图像！

下面的例子和消除背景的例子是完全相同的，只是有一个重要的地方有变化—当新的帧出现时，之前的帧就会不停保存。

```
// Capture video
if (video.available()) {
    // Save previous frame for motion detection!
    prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);
    video.read();
}
```

(显示的颜色也被改变成黑色和白色，并且有一些变量的名字有些不同，但是这些变化都比较小)

例 16-13：简单的运动检测

```

import processing.video.*;
// Variable for capture device
Capture video;
// Previous Frame
PImage prevFrame;
// How different must a pixel be to be a "motion" pixel
float threshold = 50;

void setup() {
    size(320,240);
    video = new Capture(this, width, height, 30);
    // Create an empty image the same size as the video
    prevFrame = createImage(video.width,video.height,RGB);
}

void draw() {
    // Capture video
    if (video.available()) {
        // Save previous frame for motion detection!!
        prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);
        prevFrame.updatePixels();
        video.read();
    }

    loadPixels();
    video.loadPixels();
    prevFrame.loadPixels();
    // Begin loop to walk through every pixel
    for (int x = 0; x < video.width; x++) {
        for (int y = 0; y < video.height; y++) {
            int loc = x + y*video.width; // Step 1, what is the 1D pixel location
            color current = video.pixels[loc]; // Step 2, what is the current color
            color previous = prevFrame.pixels[loc]; // Step 3, what is the previous color
            // Step 4, compare colors (previous vs. current)
            float r1 = red(current); float g1 = green(current); float b1 = blue(current);
            float r2 = red(previous); float g2 = green(previous); float b2 = blue(previous);
            float diff = dist(r1,g1,b1,r2,g2,b2);
            // Step 5, How different are the colors?
            if (diff > threshold) {
                // If motion, display black
                pixels[loc] = color(0);
            } else {
                // If not, display white
                pixels[loc] = color(255);
            }
        }
    }
    updatePixels();
}

```

如果我们只想知道在一个房间中的“整体”动作呢？在第16.5章，我们会用图像每个像素的亮度总和除以它的像素数计算出图像的平均亮度。

平均亮度 = 亮度总和 / 像素总和

我们可以用同样的方式计算平均运动量

平均运动量 = 运动量总和 / 像素总和

下面的例子显示了基于运动量而改变颜色和尺寸的圆圈。请再次注意，你不需要为了分析视频而显示它！

例 16-14：整体运动量

```
import processing.video.*;
// Variable for capture device
Capture video;
// Previous Frame
PImage prevFrame;
// How different must a pixel be to be a "motion" pixel
float threshold = 50;

void setup() {
    size(320,240);
    // Using the default capture device
    video = new Capture(this, width, height, 15);
    // Create an empty image the same size as the video
    prevFrame = createImage(video.width,video.height,RGB);
}

void draw() {
    background(0);
    // If you want to display the video
    // You don't need to display it to analyze it!
    image(video,0,0);
    // Capture video
    if (video.available()) {
        // Save previous frame for motion detection!!
        prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);
        prevFrame.updatePixels();
        video.read();
    }

    loadPixels();
    video.loadPixels();
    prevFrame.loadPixels();
    // Begin loop to walk through every pixel
    // Start with a total of 0
    float totalMotion = 0;
    // Sum the brightness of each pixel
    for (int i = 0; i < video.pixels.length; i++) {
        color current = video.pixels[i];
        // Step 2, what is the current color
        color previous = prevFrame.pixels[i];
        // Step 3, what is the previous color
        // Step 4, compare colors (previous vs. current)
        float r1 = red(current); float g1 = green(current);
        float b1 = blue(current);
```

```

float r2 = red(previous); float g2 = green(previous);
float b2 = blue(previous);

float diff = dist(r1,g1,b1,r2,g2,b2);

```

运动量是指一个像素当前颜色和之前一帧颜色之间的差异。

```

totalMotion += diff;
}

```

totalMotion是所有像素颜色差异的总和。

```

float avgMotion = totalMotion / video.pixels.length;
// Draw a circle based on average motion
smooth();
noStroke();
fill(100 + avgMotion*3,100,100);
float r = avgMotion*2;
ellipse(width/2,height/2,r,r);
}

```

averageMotion是运动量总和除以像素总和的一个平均值。



练习16-7：创建一个草图来寻找运动的平均位置。你能让圆圈跟着你的手舞动吗？

16.8 计算机视觉库

在Processing中已经有好几个计算机视觉库存在了(说不定会出现更多)。这能够让你用简单的代码来控制视觉算法。使用第三方库的好处就是因为他们拥有关于计算机视觉问题的大量探索解决方案(边缘检测，斑点，运动，跟踪颜色等等)，你不需要自己把所有细节的工作全部做完！下面是现有第三方库的简要概述。

JMyron (WebCamXtra) by Josh Nimoy et al.
<http://webcamxtra.sourceforge.net/>

它有一个好处就是只需要在Windows中有VDIG，那么JMyron就能使用。它同样保函一些内置函数可以用来运行本章所学的一些任务：运动检测以及颜色跟踪。它同样还会搜索相似的像素(被称为斑点或水珠)。

LibCV by Karsten Schmidt
<http://toxi.co.uk/p5/libcv/>

和JMyron一样，在Windows中，LibCV不需要QuickTime或者WinVDIG。但是代替了我们的代码，它使用Java Media媒体框架(JMF)来从摄像头连接和捕捉图像。LibCV同样包括了一些在其他计算机视觉库中所没有的函数，例如”background learning,background subtraction,difference images,和keystoning (perspective correction).”

BlobDetection by Julien “v3ga” Gachadoat<http://www.v3ga.net/processing/BlobDetection/>

这个库的名称很明显，它就是专门用来检测图像中的斑点。斑点被定义为一个区域内高于或低于某个阈值的像素。这个库能够将任何输入的图片返回成一组斑点数组，每一个都能告诉你它的边缘点以及边界框。

16.9 沙盒

到现在为止，我们所创建的每一个草图都在网上有相对应的页面。也许你已经有一个网站来装你的Processing工作了。一旦你开始利用视频摄像头工作，就会出现一个问题。网页程序会有一些安全限制。一个小的网页程序，例如，不能连接到用户电脑的摄像头。在大多数情况下，程序不会允许连接到任何本地设备上。

这是为了电脑的安全需求，这是允许的。如果没有这个安全限制，程序员可以使用一个小程序来连接你的硬盘驱动，删掉你所有的文件，并且发email告诉他的朋友：这是一个很酷的连接！应用程序不具有安全要求。毕竟，你可以去下应用清除和格式化硬盘。但是下载和安装应用不同于弹出URL读取小程序。假设它和应用有同样的信任程度。

如果你必须将带摄像头的Processing草图在网站上发布，这里我将提供一些解决方案和建议：<http://www.learningprocessing.com/sandbox/>.



第七课项目

按照下列步骤，开发一个集成计算机视觉技术的模拟软件。

1. 设计一个没有上色的图案。这可以是一个静态的图案(如马塞克)或者是一个动态图案(如“scribbler”)或者是一个组合图案。
2. 通过JPEG图片将图案上色。
3. 用实时视频(或者是录制好的视频)的图像来代替jpg图片。
4. 使用计算机视觉技术，通过图像的属性改变图案的行为。例如，因为形状旋转导致更亮或者是因为形状飞出屏幕像素改变，等等。

利用下面所提供的空间来进行草图的设计，以及伪代码的开发。

第八课

外面的世界

17.文本

18.数据输入

19.数据流

17. 文本

“我可以将我自己所想的一些疯狂的东西用来娱乐未来的历史学家。”

— Richard Feynman

本章内容：

- 在String(字符串)对象中存储文本
- 基本的字符串功能
- 创建以及读取字体
- 显示文本

17.1 字符串来自哪里？

在第15章中，我们探索了Processing新的内置对象数据类型用来处理图像—PImage。在这一章中，我们将要介绍另一种新的数据类型，一种新的类，名为 String。

String类不是一个全新的概念。在我们打印文本到消息窗口或者从文件中读取图像之前我们就已经处理了字符串。

```
println("printing some text to the message window"); //输出一个字符串
PImage img= loadImage("filename.jpg"); //使用字符串文件名
```

虽然我们随处都在使用字符串，但是我们还并没有去充分探索发挥它们的能力。为了了解字符串的由来，让我们回忆一下类是从哪里来的？我们知道我们能够创建自己的类(Zoog,Car,等等)。我们也可以使用Processing环境中内置的类，如PImage。并且到最后，上一章节我们了解到我们还可以通过倒入第三方库来使用某些类，如Capture或者Movie。

我们在哪可以找到关于String类的文档？

为了深入了解内置变量，函数，以及类的内容，Processing reference经常会给予我们指导。虽然从技术上说它是一个Java的类，因为Java会经常使用到它，Processing同样也在reference上给予了关于它的指南文档。此外，使用它不需要import任何语句。

<http://www.processing.org/reference/String.html>

此页面只包含了部分的String类的可用方法。在SUN的Java网站能够查看到相关的完整文档：

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html>
(同样，完整的Java API网址: <http://java.sun.com/j2se/1.4.2/docs/api>).

我们现在还不打算去介绍怎么使用Java文档(在第23章中会详细介绍)，但是你可以访问上面的links去细读。

17.2 什么是字符串？

字符串其核心就是一个奇特方式排列的一组字符数组—如果我们没有String类，我们可能只能写一些这样的代码：

```
char[] sometext = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'};
```

显然，这样的工作在Processing中是非常落后的。如果使用String对象，我们就能够很简单的做到：

```
String sometext = "How do I make a String? Type characters between quotes!";
```

从上面可以看出，字符串在一整个字符列表之间是不需要逗号和引号隔开的。然而，这仅仅只是一个字符串数据。我们必须记住，字符串是一个带有函数的对象(这个你可以在reference中找到)。同样，它和PImage对象存储数据的形式很类似:copy(),loadPixels(),等等。在第18章中，我们会进一步了解String函数。不过这里有几个例子。

函数charAt()会返回给定索引字符串中的单个字符。字符串就像数组，第一个字符串的索引就是#0!



练习17-1：下面代码的结果是什么。

```
String message = "a bunch of text here.";
char c = message.charAt(3);
println(c);
```

另一个非常有用的函数就是length()。这个函数很容易与数组中的length属性混淆。当我们访问一个字符串对象的长度时，我们必须使用带括号的length,因为我们要调用的是一个函数，而不是访问一个叫length的属性。

```
String message = "This String is 34 characters long.";
println(message.length());
```



练习17-2：循环并且一次显示一次字符串中的一个字符。。

```
String message = " a bunch of text here. ";
for (int i = 0; i < _____ ); i ++ ) {
    char c = _____ ;
    println(c);
}
```

我们同样可以使用UpperCase()函数，改变字符串中字符的大小写。

在这里你可能会感到有一些奇怪，为什么我们不直接说” message.toUpperCase() ” 然后 print” message ” 变量？取而代之的是将” message.toUpperCase() ” 分配到一个新的变量取了一个不一样的名字—” uppercase ” 。

这是因为字符串一种特殊类型的对象。它是不可改变的。不可改变的对象它的数据从来就不能改变。一旦我们创建了一个字符串，它就会一直保持不变。如果我们想改变一个字符串，就必须创建一个新的。所以这里为了方便将文字转化成大写，toUpperCase()函数就返回了一个字符串副本的大写文本。

在本章节的最后的函数就是equals()。你第一个可能会想到用” == ” 符号与之比较。

```
String one = "hello";
String two = "hello";
println(one == two);
```

从技术层面讲，当” == ” 被用于对象时，它会比较每个对象的内存地址。就算每个字符串都保函相同的字符” hello ” ，如果它们时不同的对象实例，” == ” 的结果就会出错。equals()函数能够确保我们在检查2个包含完全相同字符序列的字符串对象，无论它们存储在计算机内存中的什么位置。

```
String one = "hello";
String two = "hello";
println(one.equals(two));
```

虽然上面代码返回的结果时相同的，但是为了安全起见还是用equals()比较好。有时候草图中创建的字符串内存位置会不一样，所以” == ” 不是所有时候都通用。



练习17-3：找到下列字符串数组中重复的部分。

```
String words = {"I","love","coffee","I","love","tea"};
for (int i = 0; i < _____; i++) {
    for (int j = ____; j < _____; j++) {
        if (_____) {
            println(_____ + "is a duplicate.");
        }
    }
}
```

字符串另外一个特点就是串联，可以将2个字符串进行串联。字符串串联使用”+”符号。虽然在大多数情况下这个符号代表数字之间的相加。但是在结合字符串使用时，它代表了串联。

```
String helloworld = "Hello" + "World";
```

变量也可以与字符串进行串联

```
int x = 10;
String message = "The value of x is: " + x;
```

在第15章中我们看到了一个很好的例子，那就是读取图像数组的名称编号。



练习17-4：串联下列变量变成一个字符串，并输出。。

该举行宽度为10像素，高度为12像素，位置在(100,100)。

```
float w = 10;
float h = 12;
float x = 100;
float y = 100;
String message = _____;
println(message);
```

17.3 显示文本

在第18章中我们会继续探索String类中可用的函数，如分析和操作字符串。现在我们所学到的只是已经足够让我们将精神集中到新的学习：渲染文本。

最简单的显示方式就是将字符串print到消息窗口。当你在调试程序的时候你可能都一直在接触。例如，你需要知道鼠标的水平坐标，你会写道：

```
println(mouseX);
```

或者时你需要确定某一部分的代码会执行，你可能会print一些描述性的信息。

```
println("We got here and we're printing out the mouse location!!!");
```

虽然这是用来进行调试，但是它也可能帮助我们显示文本实现我们的目标。要将文本放入屏幕中，我们必须遵守下面一系列简单步骤。

1. 通过“Tools” → “Create Font.” 来选择一个字体。这样会创建并放置字体文件道你的数据目录中。注意第三步中的字体文件名称。Processing会使用一种特殊的字体格式”vlw”，使用图像来显示每个字母。所以，你需要设置显示字体所需的大小。如图17.1所示。

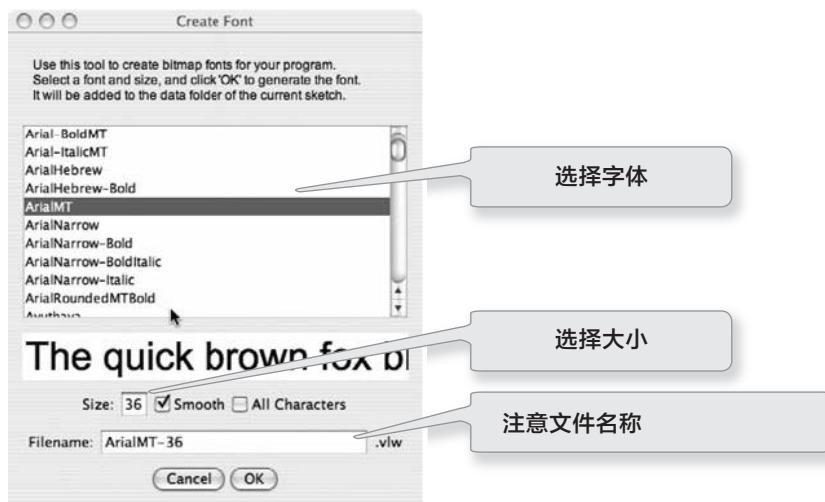


图 17.1

2. 声明一个PFont类型的对象

```
PFont f;
```

3. 在loadFont()函数中引用字体文件名称来加载字体。这个步骤只需要进行一次，通常会放在setup()中。就像加载图像一样，加载字体道内存中的这个过程是比较慢的，如果放在draw()中会严重影响草图运行的速度。

```
f = loadFont("ArialMT-16.vlw");
```

4. 指定字体使用textFont()函数。textFont()需要1个或2个参数，字体变量和字体大小，这是可选的。如果你没有标明字体的尺寸，那么字体就会按照加载时所读取的大小显示。指定字体大小会和读取时的大小不同，但是会导致文本像素化或者质量变低。

```
textFont(f,36);
```

5. 指定字体颜色使用fill()函数。

```
fill(0);
```

6. 调用text()函数用来显示文本。(这个函数就像一个形状或者图像绘制，它有三个参数—需要显示的文本，以及需要显示的文本的x和y坐标)

```
text("Mmmmm... Strings... ",10,100);
```

完整的步骤如例 17-1所示。

例 17-1：简单的显示文本

```
PFont f; // STEP 2 Declare PFont variable
void setup() {
size(200,200);
f = loadFont(" ArialMT-16.vlw "); // STEP 3 Load Font
}
void draw() {
background(255);
textFont(f,16); // STEP 4 Specify font to be used
fill(0); // STEP 5 Specify font color
text (" Mmmmm... Strings ... ",10,100); // STEP 6 Display Text
}
```

字体同样也可以使用createFont()函数。

```
myFont = createFont("Georgia", 24, true);
```

creatFont()的参数包括字体名称，
字体大小，和一个布尔值(确定是否使
用平滑抗锯齿效果)。

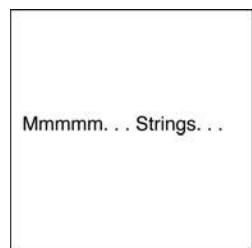


图 17.2

createFont()函数允许你使用已经安装在你计算机本地的字体。但是不存在将其变成Processing字体的选项。此外，createFont()允许字体缩放道任何尺寸而不存在模糊或者出现马塞克。更多关于createFont()的内容，请访问Processing reference: http://processing.org/reference/createFont_.html。你能通过“PFont.list()”看到所有可用的字体。

```
println("PFont.list()");
```

Print出所有createFont()可以用的字体在消息窗口中。

作为演示，createFont()将被用于所有例子。



17.4 文本动画

现在我们已经了解了需要显示文本的所有步骤，我们可以利用这本书中的其他概念来实现一个实时的动态文本。

在上手之前，让我们来看看Processing中2个比较有用的关于显示文本的函数：

`textAlign()`—指定文本靠左，靠右或者是居中对齐

例 17-2：文本对齐

```
PFont f;

void setup() {
  size(400,200);
  f = createFont("Arial",16,true);
}

void draw() {
  background(255);
  stroke(175);
  line(width/2,0,width/2,height);
  textAlign(f);
  fill(0);
```

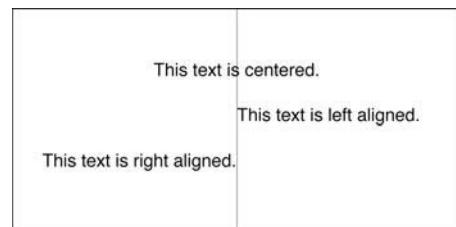


图 17.3

```

textAlign(CENTER);
text( " This text is centered. ",width/2,60);

textAlign (LEFT) ;
text( " This text is left aligned. ",width/2,100);

textAlign(RIGHT);
text( " This text is right aligned. ",width/2,140);
}

```

textAglin()能够设置文本的对齐方式。它有一个参数: CENTER或者 LEFT或者RIGHT。

textWidth()—计算并且返回任何字符或者文本字符串的宽度。

比方说我们想创建一个新闻跑马灯，文字从屏幕底部从左到右滚动。当新闻标题离开窗口时，它会再次出现在左边向右滚动。如果我们知道文本开始的x坐标，并且知道文本的宽度，我们就能判断文本什么时候消失(如图17.4所示)。**textWidth()**能够告诉我们文本的宽度。

首先，我们需要声明标题，字体以及x坐标变量，并且在**setup()**中初始化它们。

```

// A headline
String headline = "New study shows computer programming lowers cholesterol.";
PFont f; // Global font variable
float x; // Horizontal location of headline

void setup() {
  f = createFont("Arial",16,true); // Loading font
  x = width; // Initializing headline off-screen to the right
}

```

draw()函数很像我们在第5章中弹跳球的例子。首先我们要在合适的位置显示文本。

```

// Display headline at x location
textFont(f,16);
textAlign(LEFT);
text(headline,x,180);

```

我们用一个速度变量来改变x的坐标(在这里给予一个负数能让文字向左移动)。

```

// Decrement x
x = x - 3;

```

现在比较难的部分来了。当我们测试一个圆是否到达屏幕左边缘的时候，我们只需要知道它的x是不是小于0即可。而对于文本，虽然它是左对齐，但是它还是会出现在屏幕上。当x小雨0减去文字的宽度(如图17.4所示)，它才会消失。当消失时，我们需要复位x让它出现在窗口右侧。

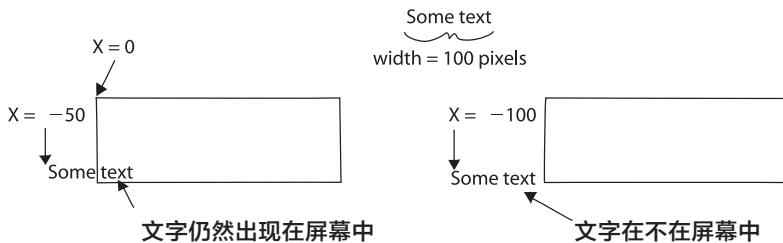


图 17.4

```
float w = textWidth(headline);
if (x < -w) {
  x = width;
}
```

如果文本的x小于这个负数宽度，那么文本就会在屏幕上完全消失

例17-3显示了一个不同变幻的标题滚动。headlines存储了一个字符串数组。

例 17-3：滚动标题

```
// An array of news headlines
String[] headlines = {
  "Processing downloads break downloading record.",
  "New study shows computer programming lowers cholesterol.",
};

PFont f; // Global font variable
float x; // Horizontal location
int index = 0;

void setup() {
  size(400,200);
  f = createFont("Arial",16,true);
  // Initialize headline offscreen
  x = width;
}

void draw() {
  background(255);
  fill(0);

  // Display headline at x location
  textAlign(LEFT);
  text(headlines[index],x,180);

  // Decrement x
  x = x - 3;
}
```

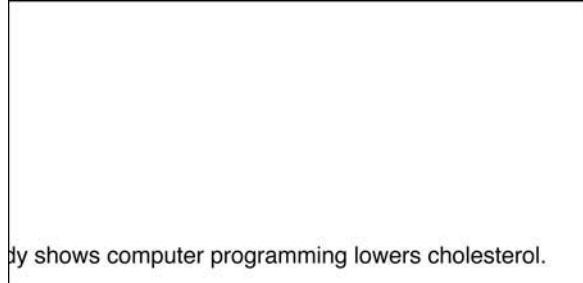


图 17.5

通过变量”index”的值，会显示字符串数组中指定的字符串。

textWidth()用于计算当前字符串的宽度。

当当前字符串离开滚动离开屏幕时，“index”就会递增用来显示新的字符串。

```
} }
```

除了textAlign()和textWidth(),Processing同样也提供textLeading(),textMode()和textSize()等其他的显示函数。这些函数没有覆盖到这个章节，如果你想进一步探索，你可以在Processing reference去学习更多。

练习17-6：创建一个股票报价机并且循环报价。从右到左滚动不断显示新的股价。



17.5 文字马赛克

结合我们在第15章和第16章所学到的关于像素数组的知识，我们可以使用图像中的像素创建字符马赛克。这是一个第16章模拟代码的一个扩展。(注意例17-4，新的文本相关的代码用的是粗体)如图17.6所示。



例 17-4：文本模拟

```

import processing.video.*;

// Size of each cell in the grid, ratio of window size to video size
int videoScale = 14;
// Number of columns and rows in our system
int cols, rows;
// Variable to hold onto capture object
Capture video;

// A String and Font
String chars = "helloworld";
PFont f;

void setup() {
    size(640,480);
    //set up columns and rows
    cols = width/videoScale;
    rows = height/videoScale;
    video = new Capture(this,cols,rows,15);

    // Load the font
    f = loadFont ("Courier-Bold-20.vlw");
}

```

```

void draw() {
    background(0);
    // Read image from the camera
    if (video.available()) {
        video.read();
    }

    video.loadPixels();

    // Use a variable to count through chars in String
    int charcount = 0;

    // Begin loop for rows
    for (int j = 0; j < rows; j + + ) {
        // Begin loop for columns
        for (int i = 0; i < cols; i + + ) {
            // Where are we, pixel-wise?
            int x = i*videoScale;
            int y = j*videoScale;

            // Looking up the appropriate color in the pixel array
            color c = video.pixels[i + j*video.width];

            // Displaying an individual character from the String
            // Instead of a rectangle
            textFont(f);
            fill(c);
            text(chars.charAt(charcount),x,y);
            // Go on to the next character
            charcount = (charcount + 1) % chars.length();
        }
    }
}

```

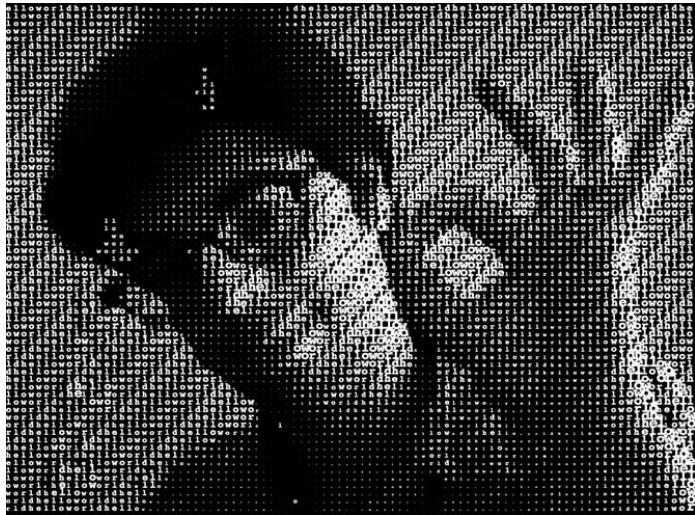
使用文本作为我们的马塞克图案。
更长的字符串可能会产生更有趣的结果。

使用一个”fixed-width(固定宽度)”
。在大多数字体中，每个字符都有不同的
宽度。而固定字体，所有的字符宽度都是
相同的。这是非常有用的，因为在这里我
们要使每个字之间显示相同的间隔。关于
非固定宽度字体显示文本字符，请参考第
17.7章节。

字符通过源文本相应的像素
位置显示其颜色。一个计数
器变量—“charcount”
用于在一个时间内穿过源文
本中的一个字符。

练习17-7：创建一个视频文本马赛克，让每个字母都是白色，但是每个字母的大小由其亮度决定。越量的地方字母就越大。这里使像素循环的一点代码(有一些需要填空)，帮助你开始。

```
float b = brightness(video.pixels[i + j*video.width]);
float fontSize = ____ * (____ / ____);
textSize(fontSize);
```



17.6 文本旋转

平移和旋转(如在第14章中所看到的)同样也可以适用于文本。例如，围绕文本中心旋转，平移到原点，并且在显示文本之前使用textAlign(CENTER)。

例 17-5：滚动标题

```
PFont f;
String message = "this text is spinning";
float theta;

void setup() {
    size(200,200);
    f = createFont("Arial",20,true);
}

void draw() {
    background(255);
    fill(0);
    textAlign(f); // Set the font
    translate(width/2,height/2); // Translate to the center
    rotate(theta); // Rotate by theta
    textAlign(CENTER);
    text(message,0,0);
    theta += 0.05; // Increase rotation
}
```



图 17.3

在平移和旋转之后文本居中对齐并且显示在(0,0)。具体参考第14章或重新复习平移和旋转。

练习17-8：让下面的文本居中对齐，并且旋转出平面的效果。让文字滚动到远方。



*A long long time ago
In a galaxy far far away*

```
String info = "A long long time ago\nIn a galaxy far far away";  
PFont f;  
float y = 0;  
  
void setup() {  
    size(400,200,P3D);  
    f = createFont("Arial",20*4,true);  
}  
  
void draw() {  
    background(255);  
    fill(0);  
    translate(_____,_____);  
    _____ (_____);  
    textAlign(CENTER);  
    text(info,_____,_____);  
    y-;  
}
```

“\n”代表“新的一行”。在Java中，不可见的字符会被纳入到“转义序列”字符串中——一个反斜杠“\”后面跟着一个字符。这里还有一些相关的
\n——新的一行
\r——回车
\t——标签
\'——单引号
\”——双引号
\\"——反斜杠

17.7 显示文本的逐个字符

在某些平面应用中，单独的渲染文本中的每个字符是有时会需要的。例如，如果每一个字符都需要有独立的移动，一般的文本不能这么做。

解决方式就是通过整个字符串循环，在每一次显示一个字符。

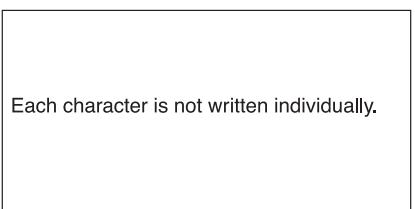
让我们看一次显示整个文本的例子。如图17.8所示。

```
PFont f;
String message = " Each character is not
written individually. ";

void setup() {
    size(400,200);
    f = createFont(" Arial ",20,true);
}

void draw() {
    background(255);
    fill(0);
    textAlign(f);
    text(message,10,height/2); }
```

一次显示文本块的所有内容使用text()。



Each character is not written individually.

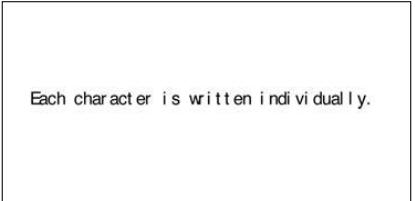
图 17.8

我们重写代码显示文本中的单独的字符，使用charAt()函数。如图17.9所示。

```
int x = 10;
for (int i = 0; i < message.length(); i++) {
    text(message.charAt(i),x,height/2);
    x += 10; }
```

所有字符之间的
间距都为10

使用charAt()函数，每个
字符每次只显示一个字符。



Each character is written individually.

图 17.9 (注意字间距是不正确的)

为每个字符调用text()函数能够让我们在以后的例子中更灵活(颜色，大小以及在字符串中放置字符)。但是在这个例子中一个非常重大的缺陷。x递增的距离是10像素，每个字符之间的距离为10个像素，虽然这基本上相同，但是实际差别还是很大，因为每个字符之间的宽度不是一样的。下面的代码中利用了textWidth() 让字符之间的距离得到了适当的调整。请注意这个例子中是如何达到适当的间距的，即使是任意大小的字符。如图17.10。

```
int x = 10;
for (int i = 0; i < message.length(); i++) {
    textSize(random(12,36));
    text(message.charAt(i),x,height/2);
    x += textWidth(message.charAt(i)); }
```

textWidth()中的字符间距是正确的



Each character is written individually.

图 17.10 (注意间隔的方式，是正确的，
即使有不同大小的字符！)



练习17-9：使用 `textWidth()` 函数，用非固定宽度的字体，重做例17-4（“模拟”文本马塞克）。下面的图像使用的是 Arial 字体。

这种“逐字逐字”的方法也可以运用到字符串中字符的独立运动。下面的例子使用面向对象变成设计字符串中每一个字符对象，他们都可以以适当的间距显示并且独立运动。

例 17-6：文本拆散

```
PFont f;
String message = "click mouse to shake it up";
// An array of Letter objects
Letter[] letters;

void setup() {
  size(260,200);
  // Load the font
  f = createFont("Arial",20,true);
  textFont(f);
  // Create the array the same size as the String
  letters = new Letter[message.length()];
  // Initialize Letters at the correct x location
  int x = 16;
  for (int i = 0; i < message.length(); i++) {
    letters[i] = new Letter(x,100,message.charAt(i));
    x += textWidth(message.charAt(i));
  }
}
```

图 17.11

字母对象会初始化他们的位置，并且显示。

```

void draw() {
    background(255);
    for (int i = 0; i < letters.length; i++) {
        // Display all letters
        letters[i].display();
        // If the mouse is pressed the letters shake
        // If not, they return to their original location
        if (mousePressed) {
            letters[i].shake();
        } else {
            letters[i].home();
        }
    }
}

// A class to describe a single Letter
class Letter {

    char letter;
    // The object knows its original "home" location
    float homex,homey;
    // As well as its current location
    float x,y;

    Letter (float x_, float y_, char letter_) {
        homex = x = x_;
        homey = y = y_;
        letter = letter_;
    }

    // Display the letter
    void display() {
        fill(0);
        textAlign(LEFT);
        text(letter,x,y);
    }

    // Move the letter randomly
    void shake() {
        x += random(-2,2);
        y += random(-2,2);
    }

    // Return the letter home
    void home() {
        x = homex;
        y = homey;
    }
}

```

对象知道它自己本身的位置，同样当前(x,y)位置应该围绕屏幕移动。

在任何点，当前的位置能够利用调用 home() 函数设置回到原来的位置。

逐字逐字的方法同样允许我们显示文字沿着曲线。在我们移动字母之前，让我们先看看我们如何会志一些曲线。这个例子中使用了大量的第13章中的三角函数。

例 17-7：盒子曲线

```

PFont f;
// The radius of a circle
float r = 100;
// The width and height of the boxes
float w = 40;
float h = 40;

void setup() {
    size(320,320);
    smooth();
}

void draw() {
    background(255);

    // Start in the center and draw the circle
    translate(width /2, height /2);
    noFill();
    stroke(0);
    ellipse(0, 0, r*2, r*2);

    // 10 boxes along the curve
    int totalBoxes = 10;
    // We must keep track of our position along the curve
    float arclength = 0;

    // For every box
    for (int i = 0; i < totalBoxes; i + +) {
        // Each box is centered so we move half the width
        arclength += w/2;
        // Angle in radians is the arclength divided by the radius
        float theta = arclength /r;

        pushMatrix();
        // Polar to cartesian coordinate conversion
        translate(r*cos(theta), r*sin(theta));
        // Rotate the box
        rotate(theta);
        // Display the box
        fill(0, 100);
        rectMode(CENTER);
        rect(0 , 0 , w,h);
        popMatrix();
        // Move halfway again
        arclength += w/2;
    }
}

```

我们的曲线是一个在窗口中间以 r 为半径的圆。

我们根据盒子的宽度沿着曲线移动。

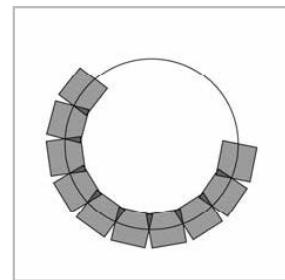


图 17.12

虽然你找到的这个数学例子很难，但是图17.12应该透露下一步。我们需要做的就是将字符从字符串中拿出来放进盒子里面。因为字符不具备同样的宽度，所以我们用一个变量”w”来保持他们的宽度不变，通过textWidth()函数我们能够给每个盒子一个合适的宽度。

例 17-7：盒子曲线

```

// The message to be displayed
String message = "text along a curve";

PFont f;
// The radius of a circle
float r = 100;

void setup() {
    size(320,320);
    f = createFont("Georgia",40,true);
    textAlign(CENTER);
    smooth();
}

void draw() {
    background(255);

    // Start in the center and draw the circle
    translate(width / 2, height / 2);
    noFill();
    stroke(0);
    ellipse(0, 0, r*2, r*2);

    // We must keep track of our position along the curve
    float arclength = 0;

    // For every box
    for (int i = 0; i < message.length(); i++) {
        // The character and its width
        char currentChar = message.charAt(i);
        float w = textWidth(currentChar);

        // Each box is centered so we move half the width
        arclength += w/2;
        // Angle in radians is the arclength divided by the radius
        // Starting on the left side of the circle by adding PI
        float theta = PI + arclength / r;

        pushMatrix();
        // Polar to cartesian coordinate conversion
        translate(r*cos(theta), r*sin(theta));
        // Rotate the box
        rotate(theta + PI/2); // rotation is offset by 90 degrees
        // Display the character
        fill(0);
        text(currentChar,0,0);
        popMatrix();
        // Move halfway again
        arclength += w/2;
    }
}

```

文本必须居中对齐

取代了固定宽度，我们需要检查每一个字符的宽度。

极点坐标系允许我们找到沿着曲线的点的位置。关于这个概念详细请回顾第13章。

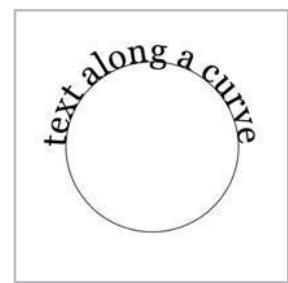
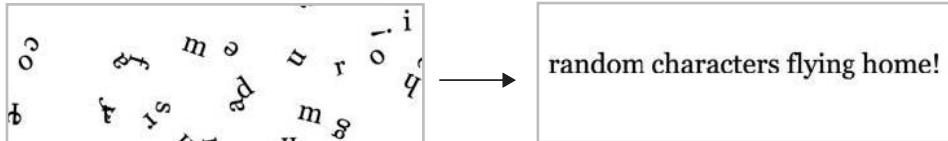


图 17.13



练习17-10：创建一个随即分散字符(带旋转的)的草图。让他们慢慢回到自己本来的位置“home”。请借鉴例 17-6 中使用面向对象编程的方法。



有一种解决方法就是使用插值法。插值是指计算2个给定数据块之间值的过程。在这个练习中，我们想知道随机的点的位置与原来本该在的位置之间的数据。插值法是一个简单的取平均值方式。把它看作一个墙，我们一直保持每次一半距离的速度行走。

```
x = (x + targetX)/2;  
y = (y + targetY)/2;
```

另一种可能就是简单的将目标的距离乘以10%。

```
x = 0.9*x + 0.1*targetX;  
y = 0.9*y + 0.1*targetY;
```

在Processing中lerp()函数能够帮你做插值。更多信息请访问：http://processing.org/reference/lerp_.html。

思考一下让你的草图中加入交互。你能用鼠标来移动字母吗？

18. 数据输入

“将一百万只猴子分给一百万台打字机。这就叫做互联网。”

— Richard Feynman

本章内容：

- 操作字符串
- 读取和写入文本文件
- HTTP请求，解析HTML
- XML,RSS订阅
- 雅虎搜索API
- 应用沙盘

本章的内容将超越显示文本和以读取和写入数据为基础的检查。我们将要学习更复杂的函数来进行字符串的操作，搜索他们，切断他们，结合他们。之后，我们将看到这些操作是如何将数据源输入的，如文本文件，web网页，XML招摇，以及第三方API，以及探索数据可视化的世界。

18.1 操作字符串

在第17章中，我们已经接触了Java函数String类的一些基本函数，如charAt(),toUpperCase(),equals(),length()。这些函数在Processing的reference页面中都有介绍。尽管如此，为了执行一些更高级的数据解析技术，我们需要探索更多一些新的字符操作函数，在Java API中有介绍(关于Java API的更多内容详见第23章)。

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html>

让我们来仔细看看下面2个字符串函数：indexOf()和substring()。

indexOf()可以定位字符串中的字符序列。它需要一个参数，一个搜索字符串，并且会返回一个数字值，这个数字值对应搜索字符串的第一次搜索。

```
String search = "def";
String toBeSearched = "abcdefghijklm";
int index = toBeSearched.indexOf(search);
```

在这个例子中index
的值是3

字符串就像一个数组，第一个字符的索引是0,最后一个字符的索引是字符串的长度减1。



练习18-1：预测下面代码的结果。

```
String sentence = "The quick brown fox jumps over the lazy dog.";
println(sentence.indexOf("quick")); _____
println(sentence.indexOf("fo")); _____
println(sentence.indexOf("The")); _____
println(sentence.indexOf("blah blah")); _____
```

如果你无法回答练习18-1的最后一行，没有问题，因为这个我们还不知道答案，所以可以看看Java reference (或者自己猜一下)。如果查找字符串不能被找到，`indexOf()`会返回1。这是一个很好的选择，因为1在String中不是一个合法的索引值，因此可以表明“没有找到”负数指数在字符串字符或数组中不存在。

在字符串中寻找一个语句之后，我们可能还想要分离字符串，并存储在不同的变量中。分离出来的字符串部分被称为子串，生成子串可以使用`substring()`函数，它需要2个参数，一个是开始索引值，一个是结束索引值。`substring()`会返回这2个索引之间的那些序列字符。

```
String alphabet = "abcdefghijklmnopqrstuvwxyz";
String sub = alphabet.substring(3,6);
```

需要注意的是子串从指定的开始索引(第一个参数)开始并且一直延伸到最后一个索引的位置(第二个参数)减1。我知道，我知道，要是只取开始索引到结束索引之间的字符不久可以了吗？虽然这在一开始可能会很方便，但是实际上最方便的还是结束索引减1.例如，如果你想创建一个子串延伸到字符串的最后一个位置，你可能会很简单的写道`string.length()`。此外此外，结束索引减1能够标记结尾，并且子串的的长度可以用结束索引减去开始索引，方便计算。



练习18-2：在“for jumps over the lazy dog”中创建子串给下面填空(不带句号)。

```
String sentence = "The quick brown fox jumps over the lazy dog.";
int foxIndex = sentence.indexOf(______);
int periodIndex = sentence.indexOf(".");
String sub = _____ . _____(_____,_____);
```

18.2 拆分和合并

在第17章中，我们看到了怎么将字符串结合在一起(简称“串联”)，使用“+”符号。让我们看一个从用户用键盘输入的字符串的串联。

例 18-1：用户输入

```

PFont f;
// Variable to store text currently being typed
String typing = "";
// Variable to store saved text when return is hit
String saved = "";

void setup() {
    size(300,200);
    f = createFont("Arial",16,true);
}

void draw() {
    background(255);
    int indent = 25;
    // Set the font and fill for text
    textAlign(fit);
    fill(0);
    // Display everything
    text("Click in this applet and type. \nHit return to save what you typed.",indent,40);
    text(typing,indent,90);
    text(saved,indent,130);
}

void keyPressed() {
    // If the return key is pressed, save the String and clear it
    if (key == '\n') {
        saved = typing;
        typing = "";
        // Otherwise, concatenate the String
    } else {
        typing = typing + key;
    }
}

```

对于键盘输入，我们使用2个变量。一个能够存储键入的文本，另一个就是存储回车键的文本。

Click in this applet and type.
Hit return to save what you typed.

44 8 15 16 23 42

图 18.1

1个字符串可以通过设置让它等于“”来清除它。

每一个被用户键入的字母都会被添加到字符串变量的结尾。



练习18-3：创建一个可以与用户聊天的草稿。例如，如果输入“cats”，草稿窗口就会出现“你喜欢猫吗？”。

Processing有2个附加函数能够让合并(或者拆分)字符串更简单。在涉及到从一个文件或者网站的数据解析中，我们会经常会得到一个字符串数组形式或一个长字符串的数据。这取决于我们想要完成的编程，知道怎样在2个储存模式之间切换是非常有用的。这里有2个新函数,split()和join()，他们会派上用场的。

“one long string or array of strings” \longleftrightarrow {"one", "long", "string", "or", "array", "of", "strings"}

让我们来看看split()函数。split()能够从字符串数组中分离一个字符组，机遇一个拆分字符被称为分隔符。它需要2个参数拆分的字符串以及分隔符。(分隔符可以是一个单独的字母或者是一个字符串)。

```
// Splitting a String based on spaces
String spaceswords = "The quick brown fox jumps over the lazy dog. ";
String[] list = split(spaceswords, " ");
for (int i = 0; i < list.length; i++) {
    println(list[i] + " " + i);
}
```

这个句号没有被设置成分隔符，因此这会
保含字符串数组中的最后一个：“dog.”。

下面是一个使用逗号作为分隔符的例子

```
// Splitting a String based on commas
String commaswords = "The,quick,brown,fox,jumps,over,the,lazy,dog. ";
String[] list = split(commaswords, ', ');
for (int i = 0; i < list.length; i++) {
    println(list[i] + " " + i);
}
```

如果你想使用多个分隔符来拆分文本，你必须使用Processing函数splitTokens()。splitTokens()的工作和split()是相同的，但是有一个例外：任何在字符串中显示的字母都可以变成分隔符。

```
// Splitting a String based on multiple delimiters
String stuff = "hats & apples, cars + phones % elephants dog .";
String[] list = splitTokens(stuff, "&, +, %");
for (int i = 0; i < list.length; i++) {
    println(list[i] + " " + i);
}
```

句号被设置成一个分隔符，因此在最后一个字符串中
不会包含句号，只有“dog”。



练习18-4：将上面的代码填写在下面，并且在Processing的消息窗口中print出来。

hats_____

如果我们在字符串中拆分数字，可以利用int()函数将数组的数字转换成一个整数数组。

```
// Calculate sum of a list of numbers in a String
String numbers = "8,67,5,309 ";
// Converting the String array to an int arr
int[] list = int(split(numbers, ', '));
int sum = 0;
for (int i = 0; i < list.length; i++) {
    sum = sum + list[i];
}
println(sum);
```

数字编码在字符串中不是一个数字，并且不
能使用数学运算，除非先将他们进行转换。

split()的反义词就是join()。join()需要一个字符串数组，并能将拆分的字符串合并成一个长的字符串。join()函数同样需要2个参数，需要合并到的数组和一个分隔符。分隔符可以是一个字母或者是一个字符串。

思考下列数组：

```
String[] lines = { " It", "was", "a", "dark", " and", "stormy", "night." };
```

使用+符号配上for循环，我们可以合并下列数组：

```
// Manual Concatenation
String onelongstring = "";
for (int i = 0; i < lines.length; i++) {
    onelongstring = onelongstring + lines[i] + " ";
}
```

我们也可以绕过这个过程以1行代码完成这个结果。

```
// Using Processing's join()
String onelongstring = join(lines, " ");
```



练习18-5：拆分下列字符串数组中的浮点数，并且计算他们的平均值。

```
String floats = " 5023.23:52.3:10.4:5.9, 901.3--2.3 ";
float[] numbers = _____(_____((_____, "_____" )));
float total = 0;
for (int i = 0; i < numbers.length; i++) {
    _____ += _____;
}
float avg = _____;
```

18.3 读取和写入文本文件

数据可以来自不同地方：网站，新闻，数据库等等。当开发一个设计数据源的应用，如数据可视化，要将程序与数据本身进行分离十非常重要的。

实际上，在定制视觉效果时，开发“虚拟”或“假”数据是非常有用的。这样能够一步一个脚印的去做我们需要做的事，当你一旦开发了虚拟的数据，然后你就可以将重点转移到如何获取实际数据来源上了。

在本节中，我们将遵循这个模式，利用最简单的数据检索方式：从文本文件中读取数据。文本文件可以被当作一个非常简单的数据库(我们可以存储程序的设置，分数列表，数字模拟等等)，或者事一个更复杂的数据源。

为了创建文本文件，你可以使用任何简单的文本编辑器。Windows中的记事本或者Mac OS X中的文本编辑器都可以应用，只需要你确定你文件的格式为“纯文本”，同样文件的名称通常为“.txt”扩展名，这样做可以避免与其他文件的混淆。就像在第15章中图像放置的位置一样，这些文本文件也应该放置在“data”目录下面，以方便Processing草稿直接使用。

一旦文本文件放入正确的文件夹，Processing的loadStrings()函数就可以用来读取文件中的内容并作为一个字符串数组。在文本文件中的每一行文字(如图18.2所示)在Processing中会对应字符串数组的每一个元素。

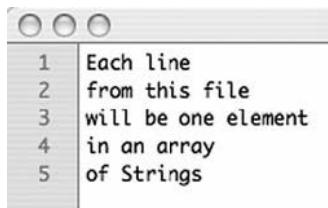


图 18.2

```
String[] lines = loadStrings("file.txt");
println("there are " + lines.length + " lines");
for (int i=0; i < lines.length; i++) {
    println(lines[i]);
}
```

这段代码会显示出来自文本文件的所有行数。如图18.2所示。

运行代码，创建一个“file.txt”的文本文件，并键入一些行文字，并把文件放入你的草稿数据目录中。



练习18-6：重写例17-3，让它从文本文件中读取标题。

数据文件中的文本能够用于生成一个简单的数据可视化。在例18-2中，读取数据文件如图18.3所示。这个数据的可视化结果如图18.4所示。

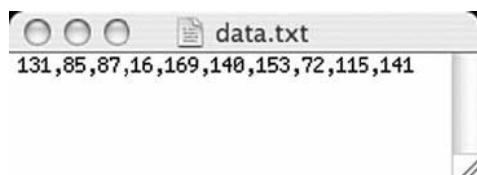


图 18.3 “data.txt”中的内容

例 18-2：利用逗号分隔文本文件中的数字

```
int[] data;

void setup() {
    size(200,200);
    // Load text file as a String
    String[] stuff = loadStrings("data.txt");
    // Convert string into an array of integers
    // using ',' as a delimiter
    data = int(split(stuff[0], ','));
}
```

文本文件中的文本被加载到一个数组中。这个数组只有一个元素，因为这个文件只有一行。然后我们可以将这个元素拆分成一个整数数组。

```
void draw() {
    background(255);
    stroke(0);
    for (int i = 0; i < data.length; i++) {
        fill(data[i]);
        rect(i*20,0,20,data[i]);
    }
    noLoop();
}
```

这个整数数组用于设置每个矩形的颜色以及高度。

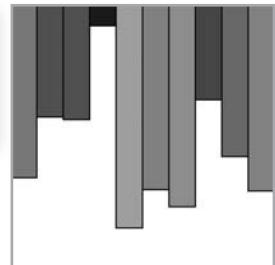


图 18.4

我们同样也可以通过文本文件的数据传递到函数构造器来使对象初始化。在例18-3中，文本文件有10行文本，每一行都代表了一个对象实例，这些值用逗号进行分隔。如图18.5所示。

例 18-3：来自文本文件的创建对象

```
Bubble[] bubbles;
void setup() {
    size(200,200);
    smooth();
    // Load text file as an array of String
    String[] data = loadStrings("data.txt");
    bubbles = new Bubble[data.length];
    for (int i = 0; i < bubbles.length; i++) {
        float[] values = float(split(data[i], ','));
        bubbles[i] = new Bubble(values[0],values[1],values[2]);
    }
}

void draw() {
    background(100);
    // Display and move all bubbles
    for (int i = 0; i < bubbles.length; i++) {
        bubbles[i].display();
        bubbles[i].drift();
    }
}
```

Bubble(气泡)对象数组的尺寸由文本中每一行数字的总和来确定。

每一行都被分割成一个浮点数组组。

这些数组中的值会传递到Bubble类的函数构造器中。

```
// A Class to describe a "Bubble"
class Bubble {
    float x,y;
    float diameter;
    float speed;
    float r,g;
```

```

// The constructor initializes color and size
// Location is filled randomly
Bubble(float r_, float g_, float diameter_) {
    x = random(width);
    y = height;
    r = r_;
    g = g_;
    diameter = diameter_;
}

// Display the Bubble
void display() {
    stroke(255);
    fill(r,g,255,150);
    ellipse(x,y,diameter,diameter);
}

// Move the bubble
void drift() {
    y += random(-3,-0.1);
    x += random(-1,1);
    if (y < -diameter*2) {
        y = height + diameter*2;
    }
}
}

```

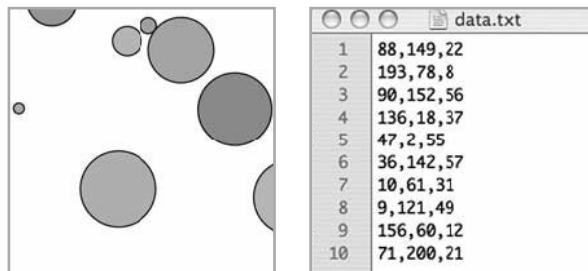
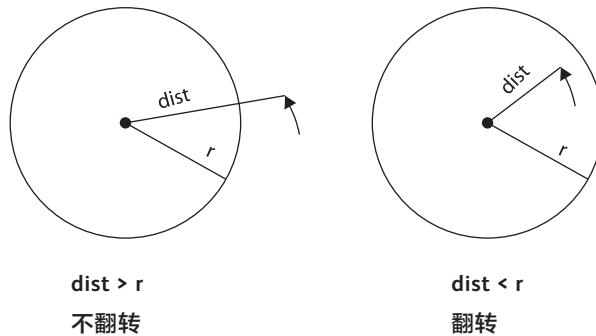


图 18.5

现在我们能够有一些很合适想法从文本文件加载信息初始化Processing的草稿，我们准备问下面的问题：如果我们想保存现有信息能够在下一次编程的时候能够加载应该怎么做？例如，我们想修改例18-3，让气泡能够随着鼠标翻转。(我们之前在练习5-5中用矩形进行了翻转，并且在例9-11中利用圆进行翻转)如图18.6所示。

图 18.6



```

boolean rollover(int mx, int my) {
    if (dist(mx,my,x,y) < diameter/2) {
        return true;
    } else {
        return false;
    }
}

```

rollover()函数在Bubble类中返回一个布尔值(true或false)，这取决于参数(mx,my)是否在规定的圆圈范围内。

rollover()函数会检查给定的点(mx,my)和气泡的点(x,y)之间的距离是否小于圆圈的半径；半径被定义为直径除以2(如图18.6所示)。如果是true，点(mx,my)就会在圆圈里面。调用这个函数要将鼠标的坐标作为参数，让我们进行测试。

```
for (int i = 0; i < bubbles.length; i++) {
    bubbles[i].display();
    bubbles[i].drift();
    if (bubbles[i].rollover(mouseX, mouseY)) {
        bubbles[i].change();
    }
}
```

一旦我们完成了change()函数来调整气泡的变量，我们就能利用Processing函数saveString()存储新的信息到文本文件中去了。saveString()和loadString()在本质上来说是相反的，它接受文件名称，字符串数组以及储存这些数组。

```
String[] stuff = { " Each String ", "will be saved ", "on a ", "separate line " };
saveStrings("data.txt", stuff);
```

这些代码就会创建如图
18.7所示的文本文件。

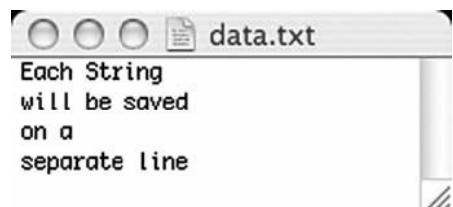


图 18.7

但是saveStrings()不会将存储的文本文件放在data目录下，而是放在草稿目录下。如果我们想文件被放置在data目录下，我们必须指定路径。同样，如果文件退出，它会被覆盖。

知道了这一点，我们能为气泡草稿编造一个saveData()函数，利用当前对象的属性重写“data.txt”。在这个例子中，只要我们鼠标进行点击，我们就会储存一个新的数据。

```
void saveData() {
    String[] data = new String[bubbles.length];
    for (int i = 0; i < bubbles.length; i++) {
        data[i] = bubbles[i].r + "," + bubbles[i].g + "," + bubbles[i].diameter;
    }
    saveStrings("data/data.txt", data);
}

void mousePressed() {
    saveData();
}
```

首先我们创建和气泡对象数字合计相同的字符串数组。

每一个字符串元素都通过每一个气泡对象的变量进行串联。

由于原来的数据文件被覆盖，当你再次运行草稿的时候，新的值会被读取。这里是一个完整的例子可以作为参考。如图18-18所示，saveStrings()生成的新数据：

例 18-4：读取以及存储数据

```
// An array of Bubble objects
Bubble[] bubbles;

void setup() {
    size(200,200);
    smooth();
    // Load text file as a string
    String[] data = loadStrings("data.txt");
    // Make as many objects as lines in the text file
    bubbles = new Bubble[data.length];
    // Convert values to floats and pass into Bubble constructor
    for (int i = 0; i < bubbles.length; i++) {
        float[] values = float(split(data[i], " "));
        bubbles[i] = new Bubble(values[0],values[1],values[2]);
    }
}

void draw() {
    background(255);
    // Display and move all bubbles
    for (int i = 0; i < bubbles.length; i++) {
        bubbles[i].display();
        bubbles[i].drift();
        // Change bubbles if mouse rolls over
        if (bubbles[i].rollover(mouseX,mouseY)) {
            bubbles[i].change();
        }
    }
}

// Save new Bubble data when mouse is Pre
void mousePressed() {
    saveData();
}

void saveData() {
    // For each Bubble make one String to be saved
    String[] data = new String[bubbles.length];
    for (int i = 0; i < bubbles.length; i++) {
        // Concatenate bubble variables
        data[i] = bubbles[i].r + "," + bubbles[i].g + "," + bubbles[i].diameter;
    }
    // Save to File
    saveStrings("data/data.txt",data);
}

// A Bubble class
class Bubble {
    float x,y;
    float diameter;
    float speed;
    float r,g;

    Bubble(float r_,float g_, float diameter_) {
        x = random(width);
        y = height;
        r = r_;
    }
}
```

气泡的数据在setup()中被加载。

气泡的数据在mousePressed()
中保存。

同一文件会被覆盖，并为
saveStrings()添加“data”指定的
文件夹路径，如图18.8所示。

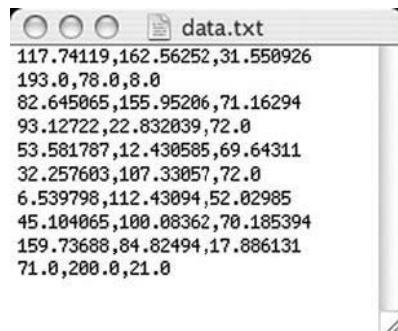


图 18.8 新的数据在存储后生成

```

g = g_-;
}

// True or False if point is inside circle
boolean rollover(int mx, int my) {
    if (dist(mx,my,x,y) < diameter/2) {
        return true;
    } else {
        return false;
    }
}

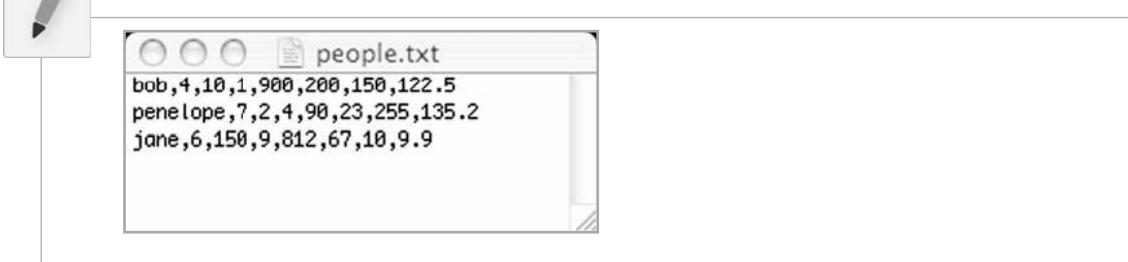
// Change Bubble variables
void change() {
    r = constrain(r + random(-10,10),0,255);
    g = constrain(g + random(-10,10),0,255);
    diameter = constrain(diameter + random(-2,4),4,72);
}

// Display Bubble
void display() {
    stroke(0);
    fill(r,g,255,150);
    ellipse(x,y,diameter,diameter);
}

// Bubble drifts upwards
void drift() {
    y += random(-3,-0.1);
    x += random(-1,1);
    if (y < -diameter*2) {
        y = height + diameter*2;
    }
}
}

```

 练习18-7：根据下列数据创建一个可视化草稿。可以随意添加改变文本。



18.4 文本解析

现在我们能够很好的使用loadStrings()工作，将文本文件储存在本地了。我们现在就可以在想我们能不能从其他地方获取数据，如网址。

```
String[] lines = loadStrings("http://www.yahoo.com");
```

当你发送一个URL路径到loadStrings()中的时候，你会得到一个原始的HTML(Hypertext Markup Language)页面源。这和你在浏览器菜单中“查看原文件”所显示的内容是相同的。对于本节，我们不需要你掌握很深的HTML知识，但是你完全不熟悉HTML，你就需要读一读<http://en.wikipedia.org/wiki/HTML>来了解一下。

不像在文本文件中利用逗号分隔符那样，在Processing中它是一种特殊格式，存储HTML的原始数据为一个字符串数组是没有任何意义的(每个元素代表一行)。可以将数组转换成一个长字符串，让事情变得简单些。正如之前章节所提到的，我们可以通过使用join()函数来完成此操作。

```
String onelongstring = join(lines, " " );
```

当从一个网页拉取原始HTML数据时，你可能不需要所有的数据，你需要的只是一小块。也许你只在寻找天气信息，股票报价或新闻标题。我们可以用一些我们学过的高级的字符操作函数来完成这些事—indexOf(),substring()以及length()—在一大块文本中找到你所需要的数据。我们之前就看到在练习18-2中的一个例子。用下面的字符串举例说明：

```
String stuff = " Number of apples:62. Boy, do I like apples or what! ";
```

比如我想在上面的文字中取出苹果数量相关的文字。我们的算法如下：

1. 找到子串“apples”的尾部。把它当作开始。
2. 找到“apples”之后的第一个句号。把它当作结尾。
3. 在开始和结尾之间做一个字符串子串。
4. 转换字符串为一个数字(如果我们想使用它)。

“end”字符串能够通过搜索字符串索引并添加索引的长度(在这里，是8)。

变成代码，就像这样：

```
int start = stuff.indexOf(" apples: ") + 8; // STEP 1
int end = stuff.indexOf(".", start); // STEP 2
String apples = stuff.substring(start, end); // STEP 3
int apple_no = int(apples); // STEP 4
```

通过上面的代码，我们就能成功，但是我们应该仔细编写代码确定它不会出错。我们可以添加一些错误检查，并归纳成一个函数：

```
// A function that returns a substring between two substrings
String giveMeTextBetween(String s, String startTag, String endTag) {
    String found = "";
    // Find the index of the beginning tag
    int startIndex = s.indexOf(startTag);
    // If we don't find anything
    if (startIndex == -1) return "";
    // Move to the end of the beginning tag
    startIndex += startTag.length();
```

这个函数用来返回在开始字符串和结束字符串中的字符串。如果开始或结束“tag”没有被找到，函数就会返回一个空的字符串。

```
// Find the index of the end tag
int endIndex = s.indexOf(endTag, startIndex);
// If we don't find the end tag,
if (endIndex == -1) return "";
// Return the text in between
return s.substring(startIndex,endIndex);
}
```

IndexOf()同样可以使用第二个参数，一个整数。第二个参数含义：查找在指定索引中第一歌出现的字符串。我们这里用来确保结束索引在开始索引后面。

带着这种技术，我们准备链接一个网站，利用Processing来获取数据到我们草稿中。例如，我们可以从 www.nytimes.com 读取HTML源代码来查看今天的头条新闻，搜索<http://finance.yahoo.com>来查看股票报价，统计在你最喜欢的博客中有多少次“Flickr”出现，等等。但是让人觉得可怕的地方是网页之间不是相同的格式，这样就很为难工程师并且很难的解析。更何况公司会经常性的该表网页源代码，所以我所做的任何例子，虽然现在我写出来了，但是随着时间推移，我的例子就不能使用了。

对于从网络中抓取数据，XML(Extensible Markup Language) 是更可靠以及更容易被解析的。不像HTML(被设计出来主要面向人)，XML被设计出来主要面向电脑以及方便不同系统之间的书序共享。在第18.17节中我们会了解XML是如何工作的。但是对于现在来说，让我们来看看从Yahoo XML天气网站给定的编码来获取天气。关于所有Yahoo XML的信息可以在:<http://developer.yahoo.com/rss/>. 天气订阅在这：

<http://xml.weather.yahoo.com/forecastrss?p10025>

一种方法是使用Processing的XML库(这样方便阅读XML文档)来获取天气数据。但是，为了演示低等级字符串解析，作为练手，我们将使用我们的loadStrings()来手动嵌入XML源来抓取技术以及搜索信息。诚然，这是一个很愚蠢的追求，因为XML被设计出来就是为了避免这种解析方式。为了方便比较，我们会在第18.7章和第18.8章中看到2个不同XML库的使用。

<yweather:condition text="Fair" code="34" temp="88" date="Wed, 01 Aug 2007 3:51 pm EDT"/>

从上面网址所给的XML源中，我们可以看到今天的温度(在写这篇文章的时候恰好是2007年8月1日)是88F—温度“88”。

温度是随时会变化的，但是XML的格式不会变化，因此我们可以推断出用于我们搜索的启始标签应该是：

temp="

结束标签应该是：

"

即在启始标签之后的第一个引号

知道了起始标签和结束标签之后，我们就可以使用**giveMeTextBetween()**来抓取温度值了。

```

String url = "http://xml.weather.yahoo.com/forecastrss?p=10003";
String[] lines = loadStrings(url);
// Get rid of the array in order to search
// the whole page
String xml = join(lines, " ");

// Searching for temperature
String tag1 = "temp = \"";
String tag2 = "\"";
temp = int(giveMeTextBetween(xml,tag1,tag2));
println(temp);

```

在Java中一个单引号表示字符串开头或者结尾。所以我们如何在字符串中包含一个单引号呢？

答案就是通过一个“逃生”符号(我们在练习17-8中遇到过这种)我们可以使用一个反斜杠让引号包含在字符串中，引号跟在反斜杠后面。例如：

```
String q = "This String has a quote \"in it";
```

例18-5检索了Yahoo天气XML所提供的温度信息，并且显示在屏幕上。这个例子同样使用了面向对象编程，把所有字符串解析函数放到了一个WeatherGrabber类中。

例 18-5：读取以及存储数据

```

PFont f;

String[] zips = { "10003", "21209", "90210" };
int counter = 0;

// The WeatherGrabber object does the work for us!
WeatherGrabber wg;

```

```

void setup() {
  size(200,200);
  // Make a WeatherGrabber object
  wg = new WeatherGrabber(zips[counter]);
  // Tell it to request the weather
  wg.requestWeather();
  f = createFont("Georgia",16,true);
}

```

```

void draw() {
  background(255);
  textAlign(CENTER);
  fill(0);
  // Get the values to display
  String weather = wg.getWeather();
  int temp = wg.getTemp();
  // Display all the stuff we want to display
  text(zips[counter],10,160);
  text(weather,10,90);
  text(temp,10,40);
  text("Click to change zip.",10,180);
  // Draw a little thermometer based on the temperature
  stroke(0);
  fill(175);
  rect(10,50,temp*2,20);
}

```

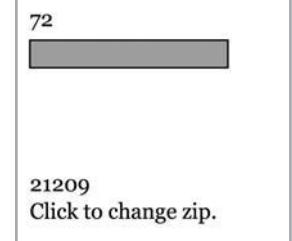


图 18.9

WeatherGrabber对象利用一个邮编进行初始化。XML数据通过调用requestWeather()函数来进行加载以及解析。

天气和温度信息从WeatherGrabber对象中获取。

```

void mousePressed() {
    // Increment the counter and get the weather at the next zip code
    counter = (counter + 1) % zips.length;
    wg.setZip(zips[counter]);
    wg.requestWeather();
}

// A WeatherGrabber class
class WeatherGrabber {
    int temperature = 0;
    String weather = "";
    String zip;

    WeatherGrabber(String tempZip) {
        zip = tempZip;
    }

    // Set a new Zip code
    void setZip(String tempZip) {
        zip = tempZip;
    }

    // Get the temperature
    int getTemp() {
        return temperature;
    }

    // Get the weather
    String getWeather() {
        return weather;
    }

    // Make the actual XML request
    void requestWeather() {
        // Get all the HTML/XML source code into an array of strings
        // (each line is one element in the array)
        String url = "http://xml.weather.yahoo.com/forecastrss?p=" + zip ;
        String[] lines = loadStrings(url);
        String xml = join(lines, " "); // Turn array into one long String
        // Searching for weather condition
        String lookfor = "<?weather:condition text = \\" ";
        String end = "\\\";
        weather = giveMeTextBetween (xml,lookfor,end);
        // Searching for temperature
        lookfor = " temp = \\" ";
        temperature = int(giveMeTextBetween (xml,lookfor,end));
    }

    // A function that returns a substring between two substrings
    String giveMeTextBetween(String s, String before, String after) {
        String found = "";
        int start = s.indexOf(before); // Find the index of the beginning tag
        if (start == -1) return ""; // If we don't find anything, send back a blank
        // String
        start += before.length(); // Move to the end of the beginning tag
        int end = s.indexOf(after,start); // Find the index of the end tag
        if (end == -1) return ""; // If we don't find the end tag, send back a blank String
        return s.substring(start,end); // Return the text in between
    }
}

```

每当鼠标点击时，都会有一个新的邮编地址
的数据请求。



练习18-8：扩展例18-5，让它能够获取明天的最高和最低温度。



练习18-9：看看Yahoo的“Word of the Day” XML的网址：<http://xml.education.yahoo.com/rss/wotd/>。使用手工解析技术获取Word of Day的信息。

18.5 文本分析

从URL中可以获得少量的文本信息。利用Processing分析大量的新闻，文章，演讲也是有可能的。Gutenberg (<http://www.gutenberg.org/>)是一个很好的项目源—这里有上千中公共领域的文本。可以利用算法分析一整本数的文本，我们先来看看一个简单的初学者例子。

例18-6中检索了莎士比亚话剧King Lear，并且使用**split Tokens()**来生成话剧里面所有的单词。草稿会一个接一个的显示这些词，还有它们各个词出现的次数。

例 18-6：King Lear数据分析

```
PFont f; // A variable to hold onto a font

String[] kinglear; // The array to hold all of the text
int counter = 0; // Where are we in the text

// We will use spaces and punctuation as delimiters
String delimiters = ".?!,: ";

void setup() {
    size(200,200);
    // Load the font
    f = loadFont("Georgia-Bold-16.vlw");

    // Load King Lear into an array of strings
    String url = "http://www.gutenberg.org/dirs/etext97/1ws3310.txt";
    String[] rawtext = loadStrings(url);
    // Join the big array together as one long string
    String everything = join(rawtext, " ");
    // Split the array into words using any delimiter
    kinglear = splitTokens(everything,delimiters);
    frameRate(5);
}

void draw() {
    background(255);

    // Pick one word from King Lear
    String theword = kinglear[counter];

    // Count how many times that word appears in King Lear
    int total = 0;
```

任何标点符号都会被作为分隔符。

King Lear的所有行文字在第一次加载的时候是一个超大的字符串，然后它会被分割成单独的单词数组。注意使用**splitTokens()**，因为我们使用空格和标点符号作为分隔符。

这个循环会计算当前单词显示出现的次数。

```

for (int i = 0; i < kinglear.length; i++) {
    if (theword.equals(kinglear[i])) {
        total++;
    }
}

// Display the text and total times the word appears
textFont(f);
fill(0);
text(theword, 10, 90);
text(total, 10, 110);
stroke(0);
fill(175);
rect(10, 50, total / 4, 20);
// Move onto the next word
counter = (counter + 1) % kinglear.length;
}

```

单词“Lear”在King Lear这个文
本中一共出现了226次

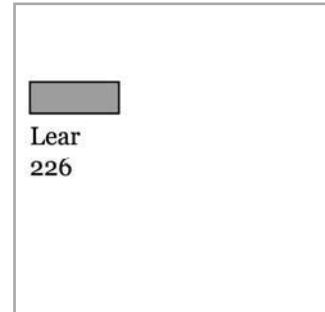
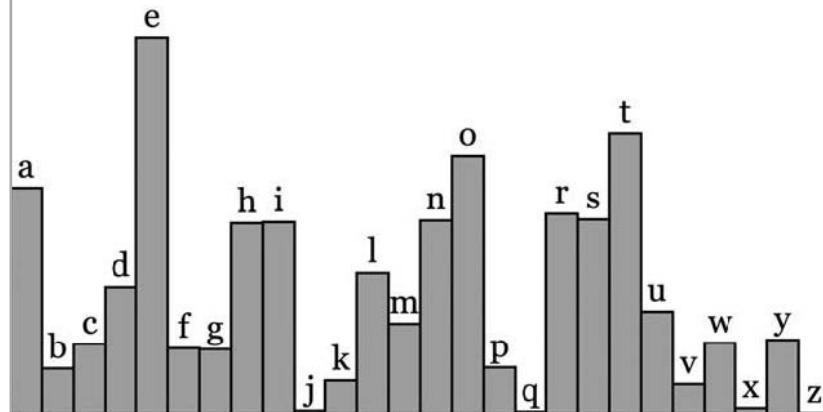


图 18.10

练习18-10：计算King Lear这本话剧集中每个字母所出现的次数，并将它们用可视化的方式显示出来。这里可视化的方式有一个（但是你可以想出更多种方式）。注意这个草稿中会需要使用charAt()函数。



18.5 异步请求

正如我们所看到的，loadStrings()函数可以从网页中获取原始数据。尽管如此，除非你的草稿在setup()中加载数据，否则你就会出现一个问题。例如，思考一个每隔5分钟在一个XML网页中抓取股票价格的草稿。每次loadStrings()被调用时，草稿就会暂停等待接受数据。任何有关的动画就会卡住。这时因为loadStrings()时一个“封闭”函数，换句话说，草稿会停下来等到loadStrings()完成它的任务，才会继续运行。对于本地文本文件来说，这个动作时非常块的。但是，这个请求对于网页(简称“HTTP 请求”)来说是异步的，这意味着web服务器会需要时间来回应你的数据请求。谁知道loadStrings()会需要多长时间呢？没有人知道，如果服务器卡你就悲剧了。

simpleML库(从这本书的网站上可以下载到:<http://www.learniningprocessing.com/simpleML>)能够通过执行这些HTTP请求同时不会暂停你Processing草稿的运行来解决这个问题，它允在你数据检索的过程中进行多任务操作以及持续连贯的动画。

这个库的函数和Processing video库的函数有点类似，在第16章中我们已经熟悉了。对于检索网页来说，你必须创建一个HTML请求对象的实例，通过引用到草稿本身(this)，和你想得到的URL请求。

```
HTMLRequest req = new HTMLRequest(this, "http://www.yahoo.com");
```

请求不会自动开始，直到你调用了makeRequest()函数时，请求才会开始。

```
req.makeRequest();
```

最后，检索数据，你必须写一个函数netEvent()。这个函数可以在调用的瞬间完成数据的请求。

这个函数是另一种回调函数，和captureEvent()(详见第16章)或者mousePressed()性质相同。在这里，是当HTML请求完成时，就会出发这个事件(或XML，我们会在某一时刻看到)。

通过函数**readRawSource()**,HTML源会将网页内容返回成一个字符串。

```
void netEvent(HTMLRequest ml) {
    String html = ml.readRawSource();
    println(html);
}
```

例18-7使用了simpleML库，会每隔10秒检测Yahoo的主页。可视化效果是任意显示的(线的颜色取决于Yahoo的HTML源的字符)。然而最重要的是要注意到动画从来没有停顿去等着数据传输。

例 18-7：利用simpleML加载URL

```
import simpleML.*;

// A Request object, from the library
HTMLRequest htmlRequest;

Timer timer = new Timer(5000);

String html = ""; // String to hold data from request
int counter = 0; // Counter to animate rectangle across window
int back = 255; // Background brightness

void setup() {
    size(200,200);
```

```

// Create and make an asynchronous request
htmlRequest = new HTMLRequest(this, "http://www.yahoo.com");
htmlRequest.makeRequest();
timer.start();
background(0);
}

void draw() {
    background(back);
    // Every 5 seconds, make a new request
    if (timer.isFinished()) {
        htmlRequest.makeRequest();
        println("Making request!");
        timer.start();
    }

    // Draw some lines with colors based on characters from data retrieved
    for (int i = 0; i < width; i++) {
        if (i < html.length()) {
            int c = html.charAt(i);
            stroke(c, 150);
            line(i, 0, i, height);
        }
    }

    // Animate rectangle and dim rectangle
    fill(255);
    noStroke();
    rect(counter, 0, 10, height);
    counter = (counter + 1) % width;
    back = constrain(back - 1, 0, 255);
}

// When a request is finished
void netEvent(HTMLRequest ml) {
    html = ml.readRawSource(); // Read the raw data
    back = 255; // Reset background
    println("Request completed!"); // Print message
}

// Timer Class from Chapter 10
class Timer {
    int savedTime;
    boolean running = false;
    int totalTime;
    Timer(int tempTotalTime) {
        totalTime = tempTotalTime;
    }

    void start() {
        running = true;
        savedTime = millis();
    }

    boolean isFinished() {
        int passedTime = millis() - savedTime;
        if (running && passedTime > totalTime) {
            running = false;
        }
    }
}

```

一个HTML请求对象，用来请求URL的源

这个请求会每5秒进行一次。这仅仅是请求，数据还没有被接收。

在数据准备好的时候netEvent()函数会自动被调用接收这些数据。

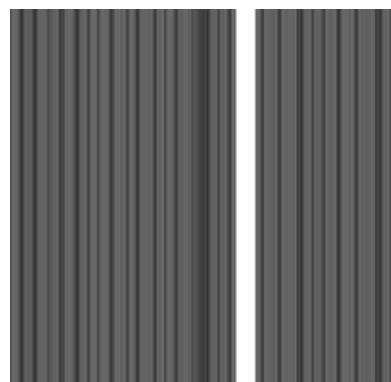


图 18.11

```
        return true;  
    } else {  
        return false;  
    }  
}
```

18.7 初级XML

第18.4节的例子展示了手写搜索独立文本数据的过程。诚然，从<http://xml.weather.yahoo.com/forecastrss?p=10003>手动解析原始的XML数据不是一个最有效的策略。但是你如果需要从一个奇怪的HTML网页来获取数据，那么手动技术是必须的。但是XML设计出来就是为了方便不同系统之间的数据分享，我们可以利用XML解析库来更方便地解析这些数据。

XML组织信息就像一个树地构造。让我们想象一个学生列表，每个学生都有单独地身份证号码，地址，e-mail以及电话号码。每一个学生地地址都有单独的城市，省，邮编。XML树状结构看起来就像图18.12所示。

XML源本身(有2个学生的列表)就是:

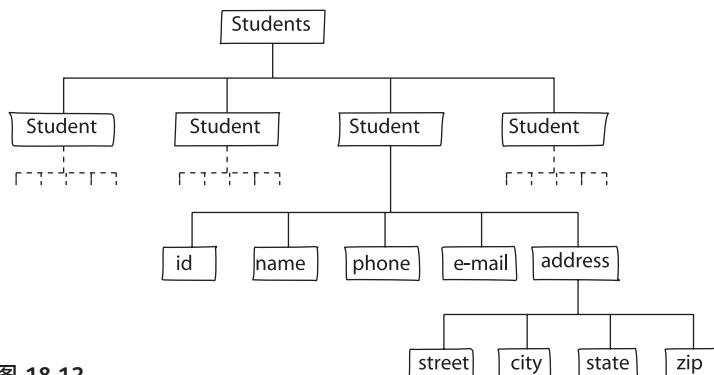


图 18.12

```
<?xml version = "1.0" encoding = "UTF-8"?>
<students>
  <student>
    <id>001</id>
    <name>Daniel Shiffman</name>
    <phone>555-555-5555</phone>
    <email>daniel@shiffman.net</email>
    <address>
      <street>123 Processing Way</street>
      <city>Loops</city>
      <state>New York</state>
      <zip>01234</zip>
    </address>
  </student>
</students>
```

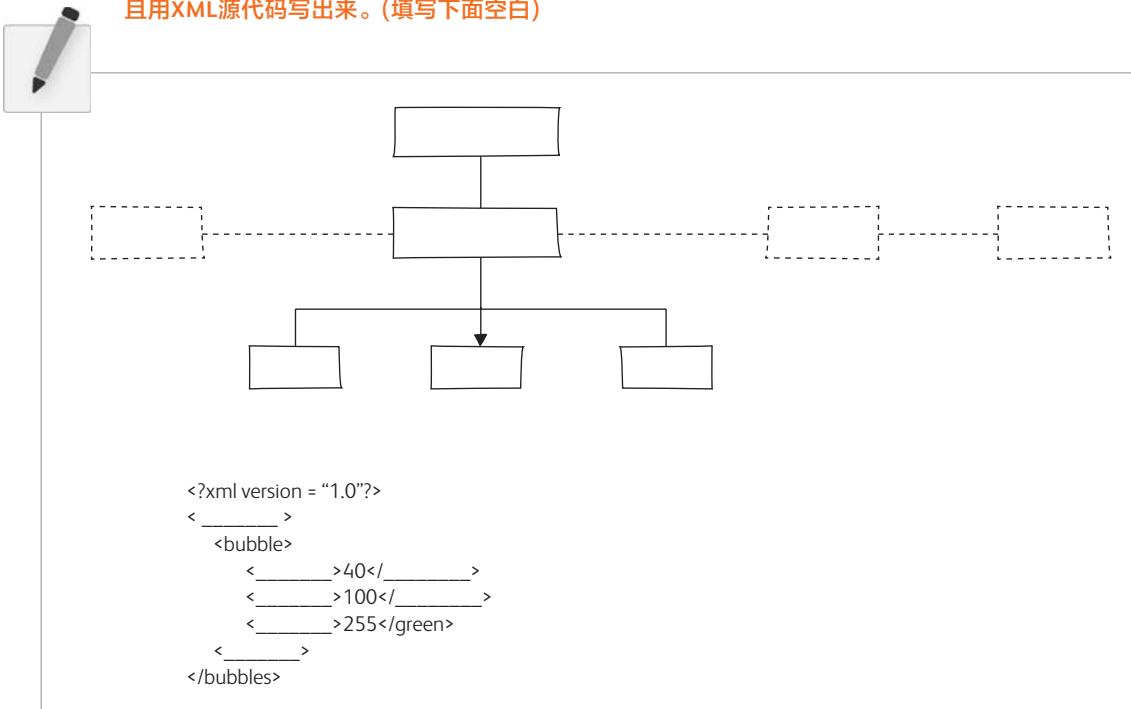
```

<student>
  <id>002</id>
  <name>Zoog </name>
  <phone>555-555-5555</phone>
  <email>zoog@planetzoron.uni</email>
  <address>
    <street>45.3 Nebula 5</street>
    <city> Boolean City</city>
    <state>Booles</state>
    <zip>12358</zip>
  </address>
</student>
</students>

```

请注意，这就是面向对象编程的相似之处。我们能够想到XML树中的条目。XML文档代表了一组学生对象的数据。每个学生对象都有一些信息，如身份证号，名称，电话号码，e-mail等等。地址也是一个对象，它又包含了一些信息，如街道，城市，省，邮编等等。

练习18-11：回顾一下例18-3中的气泡例子。为这些气泡例子设计一个XML树状结构，画在图中并且用XML源代码写出来。(填写下面空白)



回到天气的例子中，我们现在可以感觉到Yahoo的XML结构数的条目。这里是XML的源代码。(注意，为了简化起见，我已经编辑了它)

```

<?xml version = "1.0" encoding = "UTF-8" standalone = "yes"?>
<rss version = "2.0" xmlns:yweather = "http://xml.weather.yahoo.com/ns/rss/1.0">
  <channel>
    <item>
      <title>Conditions for New York, NY at 3:51 pm EST</title>
      <geo:lat>40.67</geo:lat>
      <geo:long>-73.94</geo:long>
      <link>http://xml.weather.yahoo.com/forecast/USNY0996_f.html </link>
      <pubDate>Mon, 20 Feb 2006 3:51 pm EST</pubDate>
      <yweather:condition text="Fair" code="34" temp="35" date="Mon, 20 Feb 2006 3:51 pm EST" />
      <yweather:forecast day="Mon" date="20 Feb 2006" low="25" high="37" text="Clear" code="31" />
    </item>
  </channel>
</rss>

```

这些数据变成树状构架就如图18.13所示。

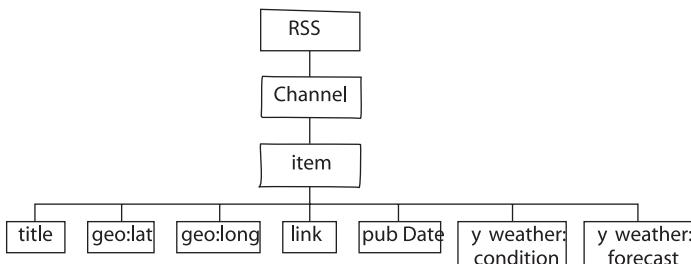


图 18.13

你可能想直到关于顶层标签“RSS”的一切。Yahoo的XML天气数据被放置在RSS格式中。RSS是“Really Simple Syndication (真正简单的整合)”的标准，并且它是XML格式对于web内容(如新闻等等)的一种标准化格式。更多关于RSS的介绍请参考百度百科：<http://baike.baidu.com/subview/1644/7031575.htm>

现在我们一个手掌形状的树状结构，我们应该先看看树状结构内部的细节。除了第一行(这一行简单的表明了这个页面是一个XML格式的页面)以外，这个XML包含了一个元素的嵌套列表，和一个开始标签chnnel以及一个结束标签/channel。一些元素内容的标签就会嵌套在其中：

```
<title>Conditions for New York, NY at 3:51 pm EST</title>
```

并且会附带一些属性(格式是属性名称=“属性值”):

```
<yweather:forecast day="Mon" date="20 Feb 2006" low="25" high="37" text="Clear" code="31" />
```

这里有一些XML解析库可以在Processing中使用，在这一章中我们会探索2个。第一个，专为这本书所设计的最简单的(但是功能最少)的库：simpleML库。simple库能够执行下面3个任务：

- 从一个XML元素中检索文本并转换成一个字符串
- 从一个XML元素的属性中检索文本并转换成一个字符串
- 从多个XML元素(具有相同的标签)中检索文本并转换成字符串数组

通过使用XMLRequest对象对XML数据进行请求。

```
XMLRequest req = new XMLRequest(this, "http://xml.weather.yahoo.com/forecastrss?p=10003");
```

同样，要等你调用了makeRequest()，请求才会开始。

```
req.makeRequest();
```

就像HTML请求一样，从XML请求中接收数据，你必须实现netEvent()，这个时候可以将XMLRequest对象作为它的参数。

函数getElementsText()会返回来自单一的XML元素(该元素的名称必须被传递到函数中)的内容。对于属性，使用getElementsAttributeText()带上这个元素的名称，然后跟着属性的名称。

例如下面的XML数据：

```
<geo:lat>40.67</geo:lat>
<yweather:forecast day="Mon" date="20 Feb 2006" low="25" high="37" text="Clear" code="31"/>
```

数据抓取代码如下：

```
void netEvent(XMLRequest ml) {
    // Getting the text of one Element
    String lat = ml.getElementText("geo:lat");

    // Getting the text of one Attribute from one Element
    String temperature = ml.getElementAttributeText("yweather:forecast","high");

    println("Latitude is: " + lat);
    println("The high temperature is:" + temperature);
}
```

函数getElementsArray()同样也可以被调用多次来检索XML元素。它会返回一个字符串数组，每个字符串对应每个XML元素。下列的例子抓取了Yahoo头条新闻XML的所有标题。

通过输入XML元素的名称(在这里，元素名称是“geo:lat”)到getElementsText()函数中，我们能够从这个XML元素中获取内容

通过输入XML元素属性的名称(在这里，元素属性名称是“yweather:forecast”)以及属性名称(这里是“high”)到getElementsAttributeText()函数中，我们能够从这个XML元素属性中获取内容

例 18-7：利用simpleML加载XML

```
import simpleML.*;
```

```
XMLRequest xmlRequest;
```

```
void setup() {
    size(200,200);
```

```

// Creating and starting the request
xmlRequest = new XMLHttpRequest(this, "http://rss.news.yahoo.com/rss/topstories");
xmlRequest.makeRequest();
}

void draw() {
    noLoop(); // Nothing to see here
}

// When the request is complete
void netEvent(XMLHttpRequest ml) {
    // Retrieving an array of all XML elements inside "<title*>" tags
    String[] headlines = ml.getElementArray("title");
    for (int i = 0; i < headlines.length; i++) {
        println(headlines[i]);
    }
}

```

使用函数**getElementArray**能够检索一个数组的XML元素。这个仅适用于XML文档中同样名称多次出现的元素。

 练习18-12：使用simpleML对Yahoo天气数据进行可视化操作。你可以用下面的代码作为你的开始。这里是XML的一些属性：`<yweather:condition text=“Fair” code=“34” temp=“35” date=“Mon, 20 Feb 2006 3:51 pm EST” /`

```

import simpleML.*;

// Variables for temperature and weather
int temperature = 0;
String weather = " ";

void setup() {
    // FILL THIS IN HOWEVER YOU LIKE!
}

void draw() {
    // FILL THIS IN HOWEVER YOU LIKE!
}

// Function that makes the weather request with a Zip Code
void getWeather(String zip) {
    String url = "http://xml.weather.yahoo.com/
    forecast?p=" + zip;
    XMLHttpRequest req = new XMLHttpRequest(this,url);
    req.makeRequest();
}

void netEvent(XMLHttpRequest ml) {
    // Get the specific XML content we want
    temperature = int(ml._("_____", "_____, "_____" ));
    weather = ml._("_____", "_____, "_____" );
}

```

18.8 使用Processing的XML库

simpleML库的函数虽然容易使用，但是范围相当有限。如果你想创建你自己的XML文档，或者利用一个自定义算法在一个文档中解析多个元素，那是很难的。对于更复杂的XML功能，这里有两种方向供你选择。第一，学习更高级的proXML (<http://www.texone.org/proxml/>)。虽然学习curve(曲线)有些艰难，但是你学习后可以直接访问XML树状结构，并且能够对XML文档进行读取和写入。

第二，学习本章我们将探索的内容，Processing中的内置XML库。

```
import processing.xml.*;
```

一旦库被倒入，第一步就是要创建一个XMLElement对象。这个对象会从XML文档(本地或网络的)中读取数据。它的函数构造器需要2个参数，“this”和XML的文件名或者URL路径。

```
String url = "xmldocument.xml";
XMLElement xml = new XMLElement(this,url);
```

和simpleML库不同，这个XML库会暂停草稿来等待文件的加载。对于XML的异步解析，你可能需要使用proXML。

一个XMLElement对象代表了XML树构架中的一个元素。当文档第一次被加载时，元素对象始终都是根元素。simpleML会穿过树构架，并找到我们需要的信息为我们工作。对于Processing的XML库来说，我们也有这些工作办法。虽然这个方法更复杂，但是我们可以有更多的控制权，能够让我们控制如何搜索以及搜索什么。

回顾图18.13，我们可以通过下面的路径顺序发现温度：

- 1.树的根元素是“RSS”
- 2.“RSS”有一个名为“Channel”的子元素
- 3.“Channel”中的第13个元素是“item”(图表只是简化了很多，只显示了Channel中的一个)
- 4.“item”中的第6个子元素是“yweather:condition”
- 5.温度被存储在“yweather:condition”中的属性“temp”

我们可以通过在getChild()函数中加入索引(从0开始，和数组一样)来访问一个元素的子元素。元素中的内容可以用getContent()进行检索，它的属性可以用getIntAttribute()，getFloatAttribute()或者getStringAttribute()进行读取。

按照上面1~5的步骤，我们可以写出代码：

```
// Accessing the first child element of the root element
XMLElement channel = xml.getChild(0);
```

```
XMLElement xml = new XMLElement(this, url);
XMLElement channel = xml.getChild(0);
XMLElement item = channel.getChild(12);
XMLElement condition = item.getChild(5);
temp = condition.getIntAttribute("temp");
```

索引中的12是指的元素中的第13个子元素

其他一些有用的函数也能调用XMLElement对象：

`getChildCount()`，返回一个XMLElement中元素的总数
`getChildren()`，返回XMLElements中的所有子元素作为一个数组

在例18-3中，我们使用了一系列逗号来分隔一个文本文件的值并将相关信息存储到气泡对象中。XML文档同样也可以以同样的方式来使用。这里是练习18-11的一种可能的解决方案，一个气泡对象的XML树。(注意这个解决方案使用了元素的属性作为红色以及绿色的填充，这不是练习18-11中提供的格式，因为我们还没有开始学习属性)

```
<?xml version = " 1.0 " ?>
<bubbles>
  <bubble>
    <diameter> 40 </diameter>
    <color red = " 75 " green = " 255 " />
  </bubble>
  <bubble>
    <diameter> 20 </diameter>
    <color red = " 255 " green = " 75 " />
  </bubble>
  <bubble>
    <diameter> 80 </diameter>
    <color red = " 100 " green = " 150 " />
  </bubble>
</bubbles>
```

这里，根元素是” bubbles”，它包含3个子元素

每个子元素“bubble”中都包含它的子元素“diameter”和“color”。“color”有2个属性，“red”和“green”

我们可以使用`getChildren()`来接收气泡元素并转化成数组。这里是例子(这里和之前使用的是相同的气泡的类)。全新的元素会加粗显示。

例 18-7：使用Processing的XML库

```
import processing.xml.*;

// An array of Bubble objects
Bubble[] bubbles;

void setup() {
  size(200,200);
  smooth();

  // Load an XML document
  XMLElement xml = new XMLElement(this, " bubbles.xml " );
  // Get the total number of bubbles
  int totalBubbles = xml.getChildCount();
  // Make an array the same size
  bubbles = new Bubble[totalBubbles];
  // Get all the child elements
  XMLElement[] children = xml.getChildren();

  for (int i = 0; i < children.length; i + + ) {
    // The diameter is child 0
    XMLElement diameterElement = children[i].getChild(0);
```

使用`getChildCount()`来获取Bubble对象的总数。

```

int diameter = int(diameterElement.getContent());
// Color is child 1
XMLElement colorElement = children[i].getChild(1);
int r = colorElement.getIntAttribute("red");
int g = colorElement.getIntAttribute("green");
// Make a new Bubble object with values from XML document
bubbles[i] = new Bubble(r,g,diameter);
}

}

void draw() {
background(100);
// Display and move all bubbles
for (int i = 0; i < bubbles.length; i++) {
    bubbles[i].display();
    bubbles[i].drift();
}
}

```

直径是第一个元素，同时红色和
绿色是第二元素。

练习18-13：使用下面的XML文档来初始化一个数组对象。设计一个对象能够用到XML中的每个元素(可任意修改XML文档)。如果你不想重写XML,你可以在这本书的网站上下载。



```

<?xml version = "1.0"?>
<blobs>
<blob>
<location x = "99" y = " 192"/>
<speed x = " 0.88238335" y = " 2.2704291"/>
<size w = "38" h = " 10"/>
</blob>
<blob>
<location x = "97" y = "14" />
<speed x = " 2.8775783" y = " 2.9483867 "/>
<size w = "81" h = "43" />
</blob>
<blob>
<location x = " 159 " y = " 193 " />
<speed x = " -1.2341062 " y = " 0.44016743 " />
<size w = "19" h = "95" />
</blob>
<blob>
<location x = " 102 " y = "53 " />
<speed x = "0.8000488" y = " -2.2791147 " />
<size w = "25" h = " 95 " />
</blob>
<blob>
<location x = "152 " y = " 181 " />
<speed x = "1.9928784 " y = " -2.9540048 "/>
<size w = "74" h = " 19 " />
</blob>
</blobs>

```

18.9 Yahoo API

在HTML和XML文档中抓取信息是非常方便的，但是对于更复杂的应用，许多网站会提供API。API(Application Programming Interface/应用程序编程接口)是一个应用程序接口，通过它可以访问其他的服务。这里有许多Processing能够使用的API，你最好看看Processing libraries网页 (<http://processing.org/reference/libraries/index.html>) 中的“Data/Protocols”章节了解更多。

在本章节中，我们会看到一个使用Yahoo API进行网络搜索的例子。虽然你可以直接访问Yahoo API,但是我已经创建了一个Processing库能够让它更简单(同样它可以执行异步搜索，不会导致草稿停顿)。你需要下载我的Processing库以及Yahoo API文件(这些被称为SDK：“Software Development Kit”/软件开发工具包)。关于说明你可以在下面网页看到：

<http://www.learningprocessing.com/libraries/yahoo/>

一旦你已经下载了你的文件，你必须首先获得一个Yahoo API密钥。在多数情况下，公司会要求你注册，然后获得密钥来访问它们的API。通过这种方式，通过这种方式，它们能够跟踪你的使用情况，确定你没有做什么恶意开发。只需要很小的代价你就能免费访问雅虎的这些功能了。你能够在这里注册ID：

<https://developer.yahoo.com/wsregapp/index.php>

一旦你有了密钥，你就可以准备开始了。库的工作原理和simpleML类似。你要创建一个YahooSearch对象，并且调用 search()函数。当搜索完成，它会转到一个事件回调:searchEvent()。在这里你可以搜索所有的信息，如URL，标题，或者摘要等等(所有可用的字符串数组)。

例 18-10：Yahoo搜索

```

import pyahoo.*;

YahooSearch yahoo;           // 创建一个YahooSearch对象。你需要将Yahoo
                             // 给你的API密钥写入其中。
void setup() {
    size(400,400);
    // Make a search object. pass in your key
    yahoo = new YahooSearch(this, "YOUR API KEY HERE ");
}

void mousePressed() {
    yahoo.search("processing.org");
}

void draw() {
    noLoop();
}

// When the search is complete
void searchEvent(YahooSearch yahoo) {
    // Get Titles and URLs
    String[] titles = yahoo.getTitles();
    String[] urls = yahoo.getUrls();
    for (int i = 0; i < titles.length; i++) {
        println("_____");
        println(titles[i]);
        println(urls[i]);
    }
}

```

搜索字符串。在默认情况下你会获得10个结果。如果你想要更多(或者更少)，你可以请求一个指定的数值：
`yahoo.search("processing.org",30);`

搜索的结果会传达到一组字符串数组中。你同样也可以通过`getSummaries()`来获得摘要。

这个库同样可以用来执行一个简单的可视化。下面的例子显示了搜索的5个名称，每一个都画一个圆(圆的尺寸与搜索到的结果的数量进行绑定)。

例 18-11：Yahoo搜索可视化

```

import pyahoo.*;

YahooSearch yahoo;
PFont f;

// The names to search, an array of "Bubble" objects
String[] names = {"Aliko", "Cleopatra", "Penelope", "Daniel", "Peter"};
Bubble[] bubbles = new Bubble[names.length];
int searchCount = 0;

void setup() {
    size(500,300);
    yahoo = new YahooSearch(this, "YOUR API HERE ");
    f = loadFont("Georgia-20.vlw");
    textFont(f);
    smooth();
    // Search for all names
    for (int i = 0; i < names.length; i++) {
        yahoo.search(names[i]);
    }
}

```

```

void draw() {
    background(255);
    // Show all bubbles
    for (int i = 0; i < searchCount; i++) {
        bubbles[i].display();
    }
}

// Searches come in one at a time
void searchEvent(YahooSearch yahoo) {
    // Total # of results for each search
    int total = yahoo.getTotalResultsAvailable();
    // Scale down the number so that it can be viewable
    float r = sqrt(total)/75;
    // Make a new bubble object
    Bubble b = new Bubble(yahoo.getSearchString(), r, 50 + searchCount*100, height/2);
    bubbles[searchCount] = b;
    searchCount++;
}

// Simple "Bubble" class to represent each search
class Bubble {
    String search;
    float x,y,r;

    Bubble(String search_, float r_, float x_, float y_) {
        search = search_;
        r = r_;
        x = x_;
        y = y_;
    }

    void display() {
        stroke(0);
        fill(0,50);
        ellipse(x, y, r, r);
        textAlign(CENTER);
        fill(0);
        text(search,x,y);
    }
}

```

`getTotalResultsAvailable()`会返回Yahoo搜索中能够搜索到你要搜索的关键词的总数。这些数字是非常庞大的，所以在使用它们作为大小之前，请缩放它们的比例。

这些搜索数据用于气泡对象的可视化。

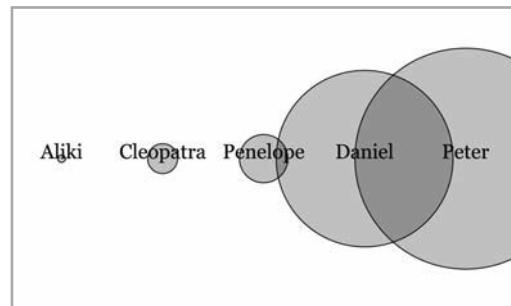


图 18.14

18.10 沙盒

当你从你的Processing开发环境中运行本地编程时，你可以自由进出网络之间。但是，当你将你的程序作为一个小应用放置在web浏览器中时，这里会有一些安全性需求，在16章中我们仅仅看到了关于连接摄像头的安全需求。

在同一域名中请求URL是允许的。例如，你的小程序存储在你的网站中你就可以很明确：

```

// Will work in the browser
String[] lines = loadStrings("http://www.myrockindomain.com/data.html");

```

但是如果你要在不同的域名中请求，你就行不通了。

```
// Will not work in the browser
String[] lines = loadStrings("http://www.yourkookoodomain.com");
```

对于这个问题的解决方法是在你的服务器上创建一个你应用的代理脚本，连接外部URL并传送信息到应用中。在本质上，你的应用会认为你只是获取本地信息。

另一个解决方式是“签署”你的应用。签署应用是一个过程，它就在说“嗨！你好，我做了这个应用。如果你相信我请说“yes”让这个应用能够正常访问需要访问的资源。”

同样，如果你只是在本地的Processing开发环境中开发草稿就不会出现这种问题。如果你需要你的应用在web服务器上运行，并且需要进行请求非服务器页面，请访问本书的网站(<http://www.learningprocessing.com/sandbox/>)去了解PHP代理脚本已经关于如何签署应用的相关内容。

19. 数据流

“I’m mad as hell and I’m not going to take this anymore!”

— Howard Beale, Network

本章内容：

- 通讯端
- 服务器
- 客户端
- 多用户处理
- 雅虎搜索API
- 串行输入

19.1 操作字符串

在第18章中，我们讨论了怎么使用`loadStrings()`对URL进行原始源的请求，可以利用simpleML库或者XML库。你提出请求，坐下来，并等待结果。当你从XML文件或网页中加载时你可能注意到这个过程不会发生在一瞬间。有时候程序会暂停几秒(甚至几分钟)。这时由于在Processing执行的后面会有一个时间长度—HTTP请求的时间。HTTP代表了“超文本传输协议”，在互联网中进行信息传输的一个请求/响应协议。

让我们想想，对于片刻来说，我们的”请求/响应”意味着什么。假设你某天早上醒来想给自己放个假，去Tuscany(地名)旅游。你会打开电脑，启动浏览器，输入`www.google.com`，然后输入`romantic getaway Tuscany villa`(浪漫的Tuscany之旅)”。你，作为客户，提出了一个请求，这就是google的工作，为客户提供响应。

客户：

只是为了介绍我自己，我是Firefox，一个浏览器，我有一个请求。我想你能够给我一些关于Tuscany度假的相关网页如果你有。

服务器：

当然，没问题，这时我的响应。这里只是一大堆的字节，但是你作为html来读取它，就能转换成一个很漂亮的有关于Tuscany度假的相关网页了！你能够让我知道你是否收到了我的响应吗？

客户：

接收到了，谢谢！

上面所描述的过程就被称作是一个异步请求以及响应，是一个服务器与客户端之间的双向通信。这个连接是暂时建立的，在其他网页进行请求的时候，这个网页会及时关闭。

顺着互联网的这种发送方式，这种技术是完全足够的。但是，你想象一下如果你在多人游戏中使用异步通信，这可能会变成一场灾难。每次一个万家发送消息到另一个玩家，连接就会打开和关闭一次，这样会造成非常大的延迟。对于在Processing中的多用户应用，我们需要的是一个实时通信，一种不同类型的连接，有一种同步被称作客户端连接。在执行实时交互的应用的元素之间，同步通信同样也是被需要的。如图19.1所示。

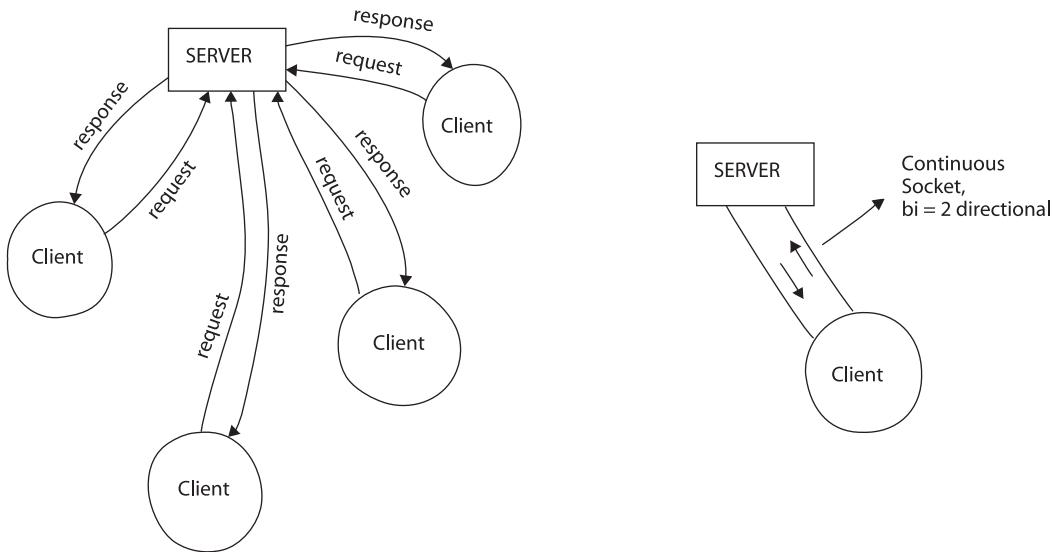


图 19.1

网络通讯连接指的是在2个程序之间通过一个网络进行持续连接。通讯端可以被用于多用户应用，如游戏，即时通讯，以及聊天(等等)。它们由一个IP地址(计算机在网络上的数字地址)和一个端口号码(一串用于从一个程序发送到另一个程序信息的数字)。

我们可以在Processing中使用通讯端来编写程序，让它们能够进行实时的通信。为了做到这一点，我们就需要使用内置的net库(processiong.net)来通过通讯端来创建服务器和客户端。

19.2 操作字符串

为了创建一个服务器，我们需要选一个端口号码。任何客户端想要连接服务器就必须知道这个号码端口号的范围从0到65,536，任何号码都是可行的，但是通常0到1,024通常是保留为公共服务器的，所以最好避免这些。如果你不确定，你可以上google搜索以下是否有这个号码在被另一个程序使用。对于我们来说，我们会使用5,204端口(这和net 库 reference里面使用的端口是相同的，你可以在processing.org中找到)。

要创建服务器，我们首先应该倒入库，并且创建一个Server对象实例。

```
import processing.net.*;
Server server;
```

服务器通过函数构造器进行初始化，它需要2个参数：“this”(可以参考第16章中的解释)和一个端口号的整数值。

```
server = new Server(this, 5204);
```

在创建的时候服务器就开始启动并等待连接。你同样可以在任何时候使用stop()函数来关闭它。

```
server.stop();
```

你可能晦记得我们在第16章的视频捕捉中所讨论的利用回调函数(captureEvent())来处理视频中新的帧。我们可以运用同样的技术找到在服务器中已经连接的客户端，使用回调函数serverEvent()。serverEvent()需要2个参数，一个服务器(用于生成事件)，一个客户端(已经连接的)。在我们检索已经连接的客户端的IP地址时我们就可能会用到这个函数。

```
// The serverEvent function is called whenever
// A new client connects
void serverEvent(Server server, Client client) {
    println("A new client has connected: " + client.ip());
}
```

当客户端发送消息(已经连接后)，serverEvent()不会有任何变化。相反的，我们必须使用available()函数来确定这个新的信息是否来自于任何客户端，客户端的广播会作为函数返回的参考，并且我们可以使用readString()函数来读取里面的内容。如果没有，就会返回值null，这意味着没有值(或者说客户端不存在)。

```
void draw() {
    // If a client is available, we will find out
    // If there is no client, it will be "null"
    Client someClient = server.available();
    // We should only proceed if the client is not null
    if (someClient != null) {
        println("Client says: " + SomeClient.readString());
    }
}
```

函数readString()是非常有用的，它在应用中够通过网络发送文本信息。如果对数据的对待方式不同，如果是作为数字(我们会在之后的例子中看到)，我们就可以使用其他的read()函数。

服务器同样可以给客户端发送信息，这里通过write()函数来实现这个功能。

```
server.write("Great, thanks for the message!\n");
```

这取决于你在做什么，通常在你信息的底部加上一个换行符号是一个很好的主意。在这里通过添加字符串“\n”来完成(有关于转义字符请详细回顾第18章)。

把上面所讲的放在一起，我们就能写出一个简单的聊天服务器。这个服务器会给任何所接受到的短语回复“How does ‘that’ make you feel?”。如例19-1所示，同如图19.2。

例 19-1：简单的服务器

```
// Import the net libraries
import processing.net.*;
// Declare a server
Server server;
// Used to indicate a new message has arrived
float newMessageColor = 255;
PFont f;
String incomingMessage = " ";
void setup() {
    size(400,200);
    // Create the Server on port 5204
    server = new Server(this, 5204);
    f = createFont(" Arial ",16,true);
}
void draw() {
    background(newMessageColor);
    // newMessageColor fades to white over time
    newMessageColor = constrain(newMessageColor + 0.3,0,255);
    textAlign(CENTER);
    fill(255);
    text(incomingMessage,width/2,height/2);
    // If a client is available, we will find out
    // If there is no client, it will be " null "
    Client client = server.available();
    // We should only proceed if the client is not null
    if (client != null) {
        // Receive the message
        incomingMessage = client.readString();
        incomingMessage = incomingMessage.trim();
        // Print to Processing message window
        println(" Client says: " + incomingMessage);
        // Write message back out (note this goes to ALL clients)
        server.write(" How does " + incomingMessage + " make you feel?\n ");
        // Reset newMessageColor to black
        newMessageColor = 0;
    }
}
// The serverEvent function is called whenever a new client connects.
void serverEvent(Server server, Client client) {
    incomingMessage = " A new client has connected: " + client.ip();
    println(incomingMessage);
    // Reset newMessageColor to black
    newMessageColor = 0;
}
```

一旦服务器开始运行，我们就能创建客户端来连接这个服务器。最终，我们会看到在Processing我们写的服务器和客户端的例子。但是这仅仅是为了验室服务器在正常空座，我们可以使用任何Telnet客户端应用来连接它。Telnet是一种标准协议，用于远程连接，并且所有的机器一般都会自带telnet功能。在Mac中已经推出了终端，在Windows中可以使用命令提示符。我还是建议使用PuTTY，一个免费的telnet客户端：<http://www.chiark.greenend.org.uk/~sgtatham/putty/>。

因为我们是连接同一个机器中的服务器，所以我们的telnet地址是localhost，意思就是本地电脑，端口是5204。我们同样可以使用地址127.0.0.1。这时为计算机上的本地程序之间的交流所保留的特殊地址(即在同一个机器上)，就相当于本地。如果我们要从不同的电脑之间进行连接，我们就必须知道运行服务器的那台电脑的IP地址。

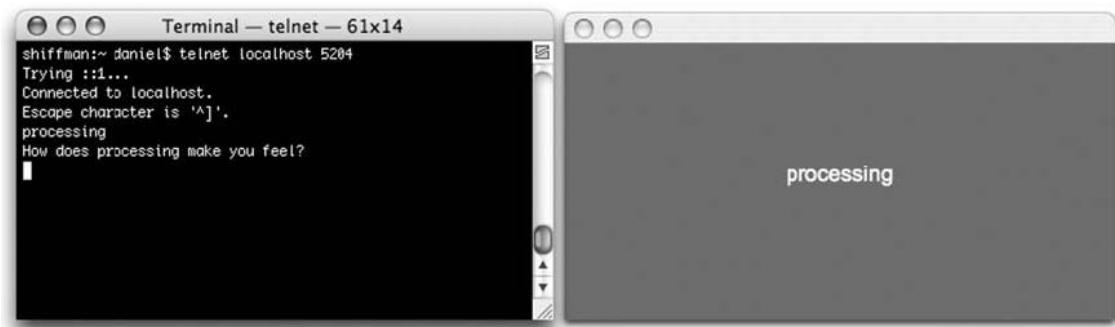


图 19.2

传统的telnet客户端会在用户按下enter之后发送消息。回车和换行都会被包括在消息中，因此当服务器消息回复时，你会注意到“How does processing” 和“make you feel”会出现在另一行。

练习19-1：使用从第15章所学习的字符串操作，修补例19-1，让它来判断如果客户发送了换行符号，服务器会在回复客户端信息之前自动移出换行符号。你需要改变“incomingMessage”变量。



```
incomingMessage = client.readString();
incomingMessage = incomingMessage._____ (_____, _____);
```

19.2 操作字符串

一旦我们编写了服务器，并且用telnet进行了测试，我们就能够在Processing中开发我们自己的客户端了。我们以和服务器相同的方式开始，倒入net库然后声明一个Client对象实例。

```
import processing.net.*;
Client client;
```

客户端的函数构造器需要3个参数—“this”，我们需要连接到的IP地址(字符串)，以及一个端口号(整数)。

```
client = new Client(this, "127.0.0.1", 5204);
```

如果服务器和客户端在不同的电脑上，我就需要知道服务器电脑的IP地址。此外，如果如果在指定的IP和端口上没有显示服务器，Processing草稿会给出错误提示：“java.net.ConnectException: Connection refused”，这意味着服务器拒绝了客户端或者服务器不存在。

使用write()函数能够很简单的把东西发送到服务器。

```
client.write("Hello!");
```

read()函数能够读取来自服务器的消息。但是read()函数每次只能读取服务器中的一个字节。所以我们需要用readString()把消息作为字符串进行读取。

在这之前我们甚至可以考虑我们可以在服务器中所读取的内容，我们必须确保有内容可读。available()函数能够检查。available()会返回等待用于读取的字节数。我们可以判断是否有内容可以读取，如果字节数大于0那么就代表有内容。

```
if (client.available() > 0) {
    String message = client.readString();
}
```

利用例18-1，(键盘输入)，我们能够创建一个Processing客户端并与服务器连接和通信，发送用户输入的消息。如例19-2所示。

例 19-2：简单的客户端

```
// Import the net libraries
import processing.net.*;

// Declare a client
Client client;

// Used to indicate a new message
float newMessageColor = 0;

// A String to hold whatever the server says
String messageFromServer = "";

// A String to hold what the user types
String typing = "";
PFont f;

void setup() {
    size(400,200);
    // Create the Client
    client = new Client(this, "127.0.0.1", 5204);
    f = createFont("Arial",16,true);
}
```

Type text and hit return to send to server.

How does processing make you feel?

图 19.3

连接到服务器的IP地址是“**127.0.0.1**
(localhost)端口号是**5024**

127.0.0.1

```

void draw() {
    background(255);

    // Display message from server
    fill(newMessageColor);
    textAlign(CENTER);
    text(messageFromServer,width/2,140);
    // Fade message from server to white
    newMessageColor = constrain(newMessageColor + 1,0,255);
    // Display Instructions
    fill(0);
    text("Type text and hit return to send to server.",width/2,60);

    // Display text typed by user
    fill(0);
    text(typing,width/2,80);

    // If there is information available to read
    if (client.available() > 0) {
        // Read it as a String
        messageFromServer = client.readString();
        // Set brightness to 0
        newMessageColor = 0;
    }
}

// Simple user keyboard input
void keyPressed() {
    // If the return key is pressed, save the String and clear it
    if (key == '\n') {
        // When the user hits enter, write the sentence out to the Server

        client.write(typing);
        typing = "";
    } else {
        typing = typing + key;
    }
}

```

通过增加亮度，新的消息会慢慢变淡

当字节数大于0的时候我们就能知道有消息会从服务器发出来。

当用户点击enter,他所输入的字符串会被传送到服务器中



练习19-2：创建一个客户端和一个服务器让它们能够互相交谈。用户从客户端发出信息，服务器会进行自动响应。例如，你能利用字符串解析结束来保存客户端发送过来的文字。客户端:How are you? 服务器: You are how?

19.4 广播

现在我们已经了解了客户端和服务器的基本工作，我们可以进行更多的网络通信的实际用途。在创阿金客户端/服务器的例子中，我们将网络发送的数据作为字符串处理，但有时候会不一样。在这个章节中，我们将着眼于写一个服务器广播数字数据发送给客户端。

这有用吗？如果你想连续广播天气或者股票价格或者摄像头运动你会怎么坐？你可以设置一个单独的电脑来运行Processing服务器来完成这个过程，并且广播这些信息。任何位置的客户端草稿都能够从这里接受到广播信息。

要演示这样的一个程序框架，我们需要编写一个服务器，广播0到255之间的数(我们一次只能发送一个字节)。然后我们会在客户端检索这些数据，最后以自己的方式来解释这些数字。

这里是服务器，他会增加一个随机数，并且广播它。

例 19-2：服务器广播一个数字(0到255之间)

```
// Import the net libraries
import processing.net.*;

// Declare a server
Server server;

PFont f;
int data = 0;

void setup() {
    size(200,200);
    // Create the Server on port 5204
    server = new Server(this, 5204);
    f = createFont("Arial", 20, true);
}

void draw() {
    background(255);
    // Display data
    textAlign(CENTER);
    fill(0);
    text(data, width/2, height/2);
    // Broadcast data
    server.write(data);
}

// Arbitrarily changing the value of data randomly
data = (data + int(random(-2,4))) % 256;
}

// The serverEvent function is called whenever a new client connects.
void serverEvent(Server server, Client client) {
    println("A new client has connected: " + client.ip());
}
```



178

图 19.4

下一步，我们将编写客户端来接受服务器的数字，并且使用数字来填充一个变量。这里的例子是假设服务器和客户端都在同一台电脑上(你可以同时打开这些例子并且在Processing中运行它)，但是在现实生活中，这种情况一般不会发生。如果你选择运行的服务器和客户端不在同一台电脑上，机器必须在本地联网中(通过路由器或者wifi)，或者互联网中。你可以找到在互联网中你的IP地址。

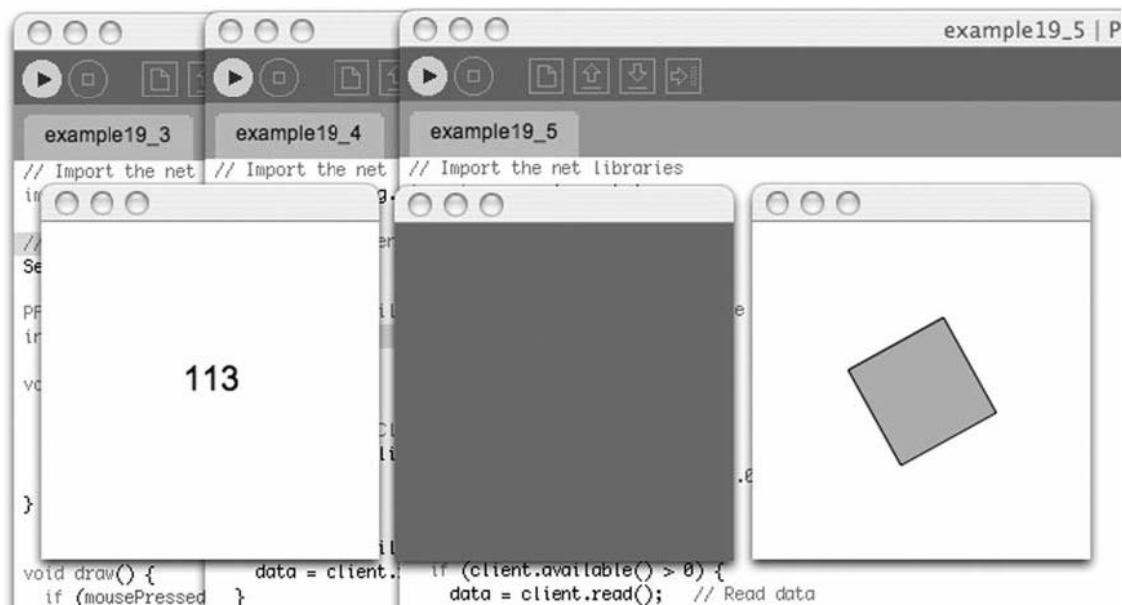


图 19.5 例19-3,19-4,19-5一起运行

例 19-4：将数字作为背景颜色的客户端

```
// Import the net libraries
import processing.net.*;

// Declare a client
Client client;

// The data we will read from the server
int data;

void setup() {
    size(200,200);
    // Create the Client
    client = new Client(this, "127.0.0.1", 5204);
}

void draw() {
    if (client.available() > 0) {
        data = client.read(); // Read data
    }
    background(data);
}
```

传入的数据被用作背景的颜色

例 19-5：将数字形状旋转角度的客户端

```
// Import the net libraries
import processing.net.*;

// Declare a client
Client client;

// The data we will read from the server
int data;

void setup() {
    size(200,200);
    smooth();
    // Create the Client
    client = new Client(this, "127.0.0.1", 5204);
}

void draw() {
    if (client.available() > 0) {
        data = client.read(); // Read data
    }
    background(255);
    stroke(0);
    fill(175);
    translate(width/2,height/2);
    float theta = (data/255.0) *
    rotate(theta);
    rectMode(CENTER);
    rect(0,0,64,64);
}
```

传入的数据被用矩形旋转的角度



练习19-3：编写一个客户端，利用服务器广播的数字来控制矩形的位置。

19.5 多用户通信，第1部分：服务器

广播的例子显示了一个服务器广播多个客户端接受，这种单向传播方式。然而这种广播方式不允许客户端回复任何信息给服务器。在这一章节中，我们将介绍如何创建多个客户端之间的多向通信的内容。

让我们来探讨以下聊天室是如何工作的。5个客户端(你和4个朋友)连接到一个服务器。一个客户端输入信息：“Hey everyone!”这个信息会发送到服务器，然后再返回给所有的5个客户端。大多数多用户应用的方式就和这一样。一个多人在线游戏，例如，客户端可能会发送玩家的相关信息以及行动到服务器中，然后服务器再广播这些数据到所有的客户端玩家中。

在Processing中多用户应用开发可以使用network库。为了演示，我们需要创建一个network，共享白板。在客户端的屏幕中拖动鼠标，草稿就会传送它的X,Y坐标到服务器中，并将这些信息在传回给其他连接着的客户端。每个连接服务器的客户端都能够看到其他人移动鼠标的动作。

除了学习多个客户端之间的通信方式，这个例子还会探索怎么发送多个值。怎么让一个客户端发送出2个值(X,Y的值)并且让服务器知道哪个是X,哪个是Y。

第一步就是为了解决客户端之间通信的开发协议。以什么格式来发送信息以及如何接受以及解释这些信息？幸运的是，关于创建，管理以及解析String对象的内容我们已经在第17和18章中学过。

假设一个客户端想发送鼠标坐标：mouseX=150, mouseY=125。我们需要将发送的信息作为一个字符串，这是为了方便翻译。有一种可能性如下所示：

“在逗号前面的数字就是X坐标，在逗号后的数字是125。在数据的结尾会出现一个星号*”。

在代码中，会如下显示：

```
String dataToSend = "100,125*";
```

或者更通常的：

```
String dataToSend = mouseX+","+mouseY+"*";
```

在这里我们已经开发了一个协议用于发送和接收数据。mouseX,mouseY的值被编译成一个字符串进行发送(数字，然后跟着逗号，然后是数字，最后是星号)。我们将会得到编译后的数据，并且会在之后解释它。我同样需要指出的是大多数例子会使用一个换行符或者回车来标记一个信息的结尾(正如我们本章的第一节所看到的)。我们使用星号有两个原因：(1)星号是清晰可见的，但是换行符不是(尤其是在书中)，(2)使用星号你可以涉及和实施任何信息协议。

我们到底发送的是什么？

数据会作为一个有序列表的字节通过网络进行发送。回顾在第4章中我们所讨论到的数据类型，1字节是一个8位的数字，也就是说，它由8个0和1组成的数字或者是一个0到255之间的值。

假设我们要发送数字42，我们可以有两种选择：

```
client.write(42); // sending the byte 42
```

在上面一行中，我们实际发送的是对应字节42

```
client.write("42"); // sending the String "42"
```

在上面一行中，我们实际发送的是一个字符串。这个字符串由2个字符组成，一个是“4”一个是“2”。我们发送了2个字节！这些字节的决定是通过ASCII (American Standard Code for Information Interchange /美国信息交换标准码)，这是一个标准化的字符编译手段。字符“A”字节对应65，字符“B”字节对应66等等。字符“4”对应字节52,字符“2”对应50。

当我们读取数据时，我们接受的数据作为是数字还是字节来进行解释，我们需要通过选择合适的read()函数来做到这一点。

```
int val=client.read(); // matches up with client.write(42);
String s=client.readString(); // matches up with client.write("42");
int num=int(s); // convert the String that is read into a number
```

我们现在已经准备好创建一个服务器来接收来自客户端的信息了。客户端的职责是将这些信息利用我们的协议进行格式化。服务器的工作仍然很简单：(1) 接收数据 (2)转发数据。这和我们在章节19.2的方法类似。

Step 1 . Receiving data.

```
Client client = server.available();
if (client != null) {
    incomingMessage = client.readStringUntil('*');
}
```

因为使用的是我们自己设计的协议，
并没有使用换行/回车来结束我们的
信息，我们需要用**readStringUntil()**
来代替**readString()**。

在这个例子中由一个新的函数readStringUntil()。readStringUntil()函数需要一个参数，一个字符。该字符会被作为传入数据的结束标志。我们只是跟随建立的协议进行发送。我们可以这么做是因为服务器和客户端同样也设计了这个协议。

一旦数据被读取，我们就准备添加：

Step 2. Relaying data back out to clients.

```
Client client = server.available();
if (client != null) {
    incomingMessage = client.readStringUntil('*');
    server.write(incomingMessage);
}
```

将信息传送给所有的客户端

下面是一个完整的服务器。当客户端连接的时候以及服务器接受到信息的时候，由消息会出现在屏幕上。

例 19-6：多用户服务器

```
// Import the net libraries
import processing.net.*;

// Declare a server
Server server;

PFont f;
String incomingMessage = " ";

void setup() {
    size(400,200);
    // Create the Server on port 5204
    server = new Server(this, 5204);
    f = createFont("Arial",20,true);
}

void draw() {
    background(255);
    // Display rectangle with new message color
    fill(0);
    textAlign(CENTER);
    text(incomingMessage,width/2,height/2);

    // If a client is available, we will find out
    // If there is no client, it will be "null"
    Client client = server.available();
    // We should only proceed if the client is not null
    if (client != null) {
        // Receive the message
        incomingMessage = client.readStringUntil('*');
        // Print to Processing message window
        println("Client says: " + incomingMessage);
        // Write message back out (note this goes to ALL clients)
        server.write(incomingMessage);
    }
}

// The serverEvent function is called whenever a new client connects.
void serverEvent(Server server, Client client) {
    incomingMessage = "A new client has connected: " + client.ip();
    println(incomingMessage);
}
```

从一个客户端受到的所有信息都会利用write()函数立即转发给所有的客户端

19.5 多用户通信，第2部分：客户端

客户端的工作有3个方面：

1. 发送mouseX和mouseY的坐标到服务器
2. 从服务器检索信息
3. 基于服务器中提供的信息显示椭圆

第一步，我们需要坚持我们建立的协议进行发送：

```
mouseX comma mouseY asterisk

String out = mouseX + "," + mouseY + "*";
client.write(out);
```

问题出现：什么时候来发送这些信息？我们可以选择将这两行代码插入到**draw()**循环中，每一帧都发送我们的鼠标坐标。在白板客户端的时候，我们只需要用户拖动鼠标发送它的坐标。

mouseDragged()函数是一个类似**mousePressed()**的事件处理函数。代替了点击鼠标函数被调用，而是当鼠标发生拖动时，就会出发事件。注意用户拖动鼠标时，函数会不断被调用。这也是我们可以选择的发送方式。

```
void mouseDragged() {
    String out = mouseX + "," + mouseY + "*";
    // Send the String to the server
    client.write(out);
    // Print a message indicating we have sent data
    println(" Sending: " + out);
}
```

将字符串和我们的协议放在一起：**mouseX逗号mouseY星号**

第二步，检索服务器的信息，这就像之前的客户端例子一样。但是唯一的区别是，我们这里用的是**readStringUntil()**，它遵循着“数字 - 逗号 - 数字 - 星号”的协议。

```
if (client.available() > 0) {
    // Read message as a String, all messages end with an asterisk
    String in = client.readStringUntil('*');
    // Print message received
    println(" Receiving: " + in);
}
```

一旦数据被放到一个字符串对象中，它就能够通过第18章所学到的解析技术来进行解释。

首先，会利用逗号(或者星号)作为分隔符，将字符串分隔到一个字符串数组中。

```
String[] splitUp = split(in, ",");
```

字符串数组会被转化成一个整数数组中(长度为2)

```
int[] vals = int(splitUp);
```

这些正是被用于显示椭圆。

```
fill(255,100);
noStroke();
ellipse(vals[0],vals[1],16,16);
```

下面是一个完整的客户端草稿。

例 19-7：多用户客户端

```
// Import the net libraries
import processing.net.*;

// Declare a client
Client client;

void setup() {
    size(200,200);
    // Create the Client
    client = new Client(this, "127.0.0.1", 5204);
    background(255);
    smooth();
}

void draw() {
    // If there is information available to read from the Server
    if (client.available() > 0) {
        // Read message as a String, all messages end with an asterisk
        String in = client.readStringUntil('*');
        // Print message received
        println("Receiving: " + in);
        // Split up the String into an array of integers
        int[] vals = int(splitTokens(in, "*"));
        // Render an ellipse based on those values
        fill(0,100);
        noStroke();
        ellipse(vals[0],vals[1],16,16);
    }
}

// Send data whenever the user drags the mouse
void mouseDragged() {
    // Put the String together with our protocol: mouseX comma mouseY asterisk
    String out = mouseX + "," + mouseY + "*";
    // Send the String to the server
    client.write(out);
    // Print a message indicating we have sent data
    println("Sending: " + out);
}
```

客户端从服务器读取信息，并且通过我们的协议进行解释，最后通过splitTokens()解析信息。

当鼠标被拖动时，信息就会被发送。注意客户端也会接收他们自己发送的信息！这里没有任何视觉化显示。

19.7 多用户通信，第3部分：合并在一起

当运行多用户应用时，元素推出的顺序是非常重要的。除非服务器草稿已经在运行，否则客户端草稿就会失败。

首先你必须明确服务器的IP地址，然后选择一个端口，并添加到服务器上，最后运行服务器。

之后你就可以利用正确的IP地址和端口来启动客户端。

如果你正工作在一个多用户项目上，你最有可能的就是在不同的电脑上运行服务器和客户端。毕竟这是多用户应用的一个首要优势。但是，作为测试和开发，所有的内容都在一个电脑上运行会很方便。在这里服务器地址会是“localhost”或者 127.0.0.1 (注意这个IP地址被使用在本章的一些例子中)。

在章节 21.3 中我们会学习到 Processing 的“export to application/导出应用”，它可以允许你为服务器导出一个标准的独立应用，这样当你在开发客户端的时候你就能后台运行服务器。关于“导出应用”如何工作的详细介绍请看第 18 章。你同样可以运行多个客户端来模拟环境。图 19.6 显示了在服务器上运行的 2 个客户端实例。

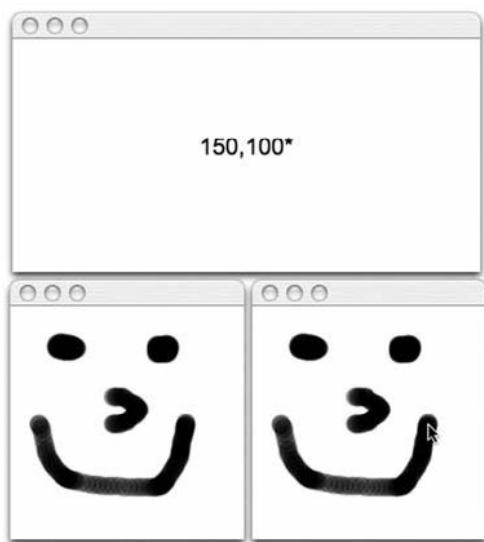


图 19.6 例 19-6, 19-7 一起运行



练习 19-4：让白板可以涂显色。每个客户端除了发送 XY 坐标还需要发送一个红色、绿色和蓝色的值。对于服务器的工作，你不需要做任何改动。



练习 19-5：创建一个 2 个玩家的网络角色扮演游戏。这是一个复杂的赋值，所以建立起来会很慢。对于实例，你应该首先在不要互联网的情况下进行工作(如果你不会，这本书的网站提供一一个例子)。你同样需要对服务器做一些修改。具体而言，服务器需要给万家分配一个奖，让他们连接(左或右)。

19.8 串行通信

学习网络通信让串行通信变成一个轻而易举的事情。串行通信涉及到从计算机串行端口读取字节。这些字节可能来自你购买的硬件(串行游戏手柄)，或者是你自己设计的一个电路板，或变成的一个微控制器。

这本书的内容没有涵盖外部硬件方面的串行通信。但是如果你感兴趣学习更多的物理计算，我建议你读一读物理计算的书：Dan O'Sullivan 写的 Sensing and Controlling the Physical World with Computers (Course Technology PTR)，Tom Igoe 写的 Practical Methods for Connecting Physical Objects (Make Books)。Arduino (<http://www.arduino.cc/>) 和 Wiring (<http://wiring.org.co/>) 网站同样都是很好的资源。Wiring 和 Arduino 都是模仿 Processing 语言开发的开源物理计算平台。我会提供一些 Arduino 相关的代码作为参考，但这本书中的材料只包含数据到达 Processing。

串行通信指的序列中发送数据的过程，每次一个字节。这是一个怎么将数据发送到我们的客户端/服务器的例子。Processing 的 serial 库设计出来用于本地设备的串行通信，大多数是通过USB (Universal Serial Bus) 端口进行通信。词语“serial/串行”指的是串行端口，被用于调制解调器的交互，新型计算机上面很少发现。

从串行端口读取数据的过程和网络客户端/服务器的读取数据过程几乎是相同的，但是有一些不同。首先，不是导入 network 库，我们需要导入 serial 库并创建 serial 对象。

```
import processing.serial.*;
Serial port = new Serial(this, "COM1", 9600);
```

串口构造器需要三个参数。第一个是 “this” 它代表了是这个小程序(详见第16章)。第二个参数是一个字符串，代表通信端口已经被使用。电脑的标签端口都有名字。在 PC 中，他们可能是 “COM1,” “COM2,” “COM3,” 等等。在基于UNIX的计算机上(例如MAC OS X)，他们通常被标记为 “/dev/tty.something” ， “something” 代表了一个终端设备。如果你正在使用USB设备，在工作之前你可能会需要安装USB驱动。关于Arduino如何工作的，请查看指南：<http://www.arduino.cc/en/Guide/HomePage>。

```
String[] portList = Serial.list();
println(portList);
```

你同样可以使用serial库中的list()函数返回一组字符串对象数组print出可用的端口。

```
String[] portList = Serial.list();
Serial port = new Serial(this, portList[0], 9600);
```

如果你将断过作为列表中的第一个，例如，你调用的构造器就看起来像：

第三个参数是数据传输的速率，通常情况下是9600 baud。

```
port.write(65); // Sending the byte 65
```

通过使用write()函数来发送字节。可以发送下列的数据类型：字节，字符，字节数组和字符串。请记住，如果你发送的是一个字符串，实际上就是在发每一个字符的原始ASCII byte值。

```
void serialEvent(Serial port) {
    int input = port.read();
    println("Raw Input: " + input);
}
```

读取数据可以使用和客户端/服务器中所用的相同的函数：read(), readString(), readStringUntil()。一个回调函数，serialEvent()，当串口事件发生时(只要有数据可被读取时)这个函数就会被触发。

例 19-8：从串口端读取数据

```
import processing.serial.*;

int val = 0; // To store data from serial port, used to color background
Serial port; // The serial port object

void setup() {
    size(200,200);
    // In case you want to see the list of available ports
    // println(Serial.list());

    // Using the first available port (might be different on your computer)
    port = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
    // Set the background
    background(val);
}

// Called whenever there is something available to read
void serialEvent(Serial port) {
    // Read the data
    val = port.read();
    // For debugging
    // println(" Raw Input: " + input);
}
```

初始化列表中第一个端口
的串口对象

串口的数据被用于填充背
景的颜色

串口数据在serialEvent()中使用read()函
数读取，并且分配一个全局变量“val”。

如果什么东西都没有能够读取到，那么read()函数会返回一个1。但是假设你已经把这个函数写进了serialEvent()，那么总会有数据可以被读取。下面是一个从串行端口读取数据并将数据填充草稿背景颜色的例子。

作为参考，如果你使用Arduino，相应的代码应该是：

```
int val;

void setup() {
    beginSerial(9600);
    pinMode(3, INPUT);
}

void loop() {
    val = analogRead(0);
    Serial.print(val,BYTE);
```

这个不是Processing代码！这是Arduino
代码。关于更多Arduino的介绍请访
问：<http://www.arduino.cc/>

19.9 带握手协议的串口通信

添加握手组建到串口通信代码中通常是非常有优势的。如果一个硬件设备发送的字节速度快于Processing草稿可以读取的速度，这就会出现草图滞后的尴尬情况。传感器的值可能会迟到，使得互动混乱或者误导用户。在发送信息的过程中“握手协议”能够缓解这种滞后。

当草稿开始时，它会发送一个字节硬件设备要求的数据。

例 19-9：握手协议

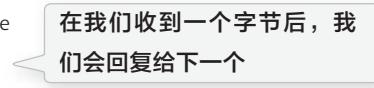
```
void setup() {
    size(200,200);
    // In case you want to see the list of available ports
    // println(Serial.list());

    // Using the first available port (might be different on your computer)
    port = new Serial(this, Serial.list()[0], 9600);
    // Request values from the hardware device
```

} 
字节65告诉串口设备我们要接收的数据。

当草稿在serialEvent()中完成这1个字节的过程滞后，它会再次寻求一个新的值。

```
// Called whenever there is something available to read
void serialEvent(Serial port) {
    // Read the data
    val = port.read();
    // For debugging
    // println("Raw Input: " + input);
```

} 
在我们收到一个字节后，我们会回复给下一个

当被请求时，只要是被设计出来的硬件设备就会发送传感器值，任何滞后都会被淘汰。这里是已经修正过的Arduino代码。这个例子不用关注需求字节是什么，只要注意关注这里是一个字节。更高级的版本会针对不同的请求有不同的回复。

```
int val;

void setup() {
    beginSerial(9600);
    pinMode(3, INPUT);
}

void loop() {
    // Only send out if something has come in
    if (Serial.available() > 0) {
        Serial.read();
        val = analogRead(0);
        Serial.print(val,BYTE);
    }
}
```

这个不是Processing代码！这是Arduino代码。更多关于Arduino的资料，请访问[http://www.arduino.cc/。](http://www.arduino.cc/)

19.10 带有字符串的串口通信

在你需要从串口中检索多个值的情况下(或者是大于 255 的数字)，`readStringUntil()` 函数会很有用。例如，假设你像读取来自3个传感器的值，并且使用这些值作为草稿的背景颜色的红、绿、蓝部分。在这里，我们将会使用和在多用户白板例子中同样的协议。我们会询问硬件要求(传感器)，要发送数据如下：

传感器值 1 逗号 传感器值 2 逗号 传感器值 3 星号

例如：

104,5,76*

例 19-10：带有字符串的串口通信

```
import processing.serial.*;

int r,g,b; // Used to color background
Serial port; // The serial port object

void setup() {
    size(200,200);
    // In case you want to see the list of available ports
    // println(Serial.list());
    // Using the first available port (might be different on your computer)
    port = new Serial(this, Serial.list()[0], 9600);

    // Request values right off the bat
    port.write(65);
}

void draw() {
    // Set the background
    background(r,g,b);
}

// Called whenever there is something available to read
void serialEvent(Serial port) {
    // Read the data
    String input = port.readStringUntil('*');
    if (input != null) {
        // Print message received
        println("Receiving: " + input);
        // Split up the String into an array of integers
        int[] vals = int(splitTokens(input, ",*"));
        // Fill r,g,b variables
        r = vals[0];
        g = vals[1];
        b = vals[2];
    }
    // When finished
    port.write(65);
}
```

串口数据在 `serialEvent()` 中使用 `readStringUntil()` 函数进行读取，并将 '*' 作为结尾字符。

数据利用逗号或者星号作为分隔符，分成了一族字符串数组，并且转换成一组整数数组。

3个全局变量被输入的数据填充。

相应的Arduino代码：

```

int sensor1 = 0;
int sensor2 = 0;
int sensor3 = 0;

void setup(){
    beginSerial(9600);
    pinMode(3, INPUT);
}

void loop(){
    if (Serial.available() > 0) { //only send if you have hear back
        Serial.read();
        sensor1 = analogRead(0);
        sensor2 = analogRead(1);
        sensor3 = analogRead(2);

        // Send the integer out as a String using "DEC"
        Serial.print(sensor1,DEC);
        // Send a comma -- ASCII code 44
        Serial.print( ",", BYTE);
        Serial.print(sensor2,DEC);
        Serial.print( ",", BYTE);
        Serial.print(sensor3,DEC);
        // Send an asterisk -- ASCII code 42
        Serial.print( "*", BYTE);
    }
}

```

这个不是Processing代码！这是Arduino代码。更多关于Arduino的资料，请访问
[http://www.arduino.cc/。](http://www.arduino.cc/)



练习19-6：如果你有一个Arduino电路板，运用你已经做好的草稿来建立属于你自己的交互控制。（在你尝试你自己的创意之前，你应该确保在本章中所提供的简单的例子能够成功运行）



第八课项目

创建一个外部加载信息(如本地文件， web网页， XML， 服务器或者串口连接)到Processing中的数据可视化。

确保你建立将木的增量。例如，先尝试设计没有真实数据的可视化效果(使用随机数或者硬代码)。如果从web中读取数据，当你在开发这个可视化项目的时候请使用一个本地文件。不要担心数据的真假问题，在连接真实数据之前，请好好完成可视化部分的设计。

实验一个抽象等级。尝试在屏幕中显示逐个逐个的显示手写的文本信息。建立一个抽象系统，让这些输入进取的数据影响到对象的行为（你甚至可以使用在第六课项目中的“生态系统”）。

使用下面所提供的空间来设计你的草稿。

第九课

制造声音

20.音频

21.导出

20. 音频

“Check. Check. Check 1. Sibilance. Sibilance. Check. Check. Check 2. Sibilance. Sibilance.”

—Barry the Roadie

本章内容：

- 关于音频的库
- 简单的音频回放
- 调整音量、音调以及声道
- 将麦克风作为音频传感器

Processing 没有内置对音频的支持。这并不意味这 Processing 抵制音频。它仅仅是是没有内置音频内容。正如我们在这本书中所介绍的，Processing 是一种编程语言和开发环境，并且基于 Java，专门用于视觉方面的设计。因此，如果你想开发关于音频的大型互动应用，你真的应该先问问自己：Processing 这种编程环境是否适合这个应用？在本章会帮助你回答这个闻听，我们会探索音频在 Processing 中工作的可能性以及局限性。

将音频加载到 Processing 草稿中能够有多种不同的方式。因为 Processing 在它的核心库中没有对音频的支持，很多 Processing 开发者选择通过第三方应用将音频加载，如 PureData (<http://www.puredata.org/>) 或者 Max/MSP (<http://www.cycling74.com/>)。Processing 能够通过 OSC (“open sound control”/开放式音频控制) 协议将电脑和多个设备之间的网络通信进行连接。这里可以使用 network 库(之前章节有提到)或者 oscP5 库 (<http://www.sojamo.de/libraries/oscP5>) 来实现。

访问 Processing 的网站同样也能看到一些有关于 Processing 对音频有帮助的库：回放音频样本，分析从麦克风输入的音频，音频合成以及发送和接受 midi 信息。这一章我们会主要聚焦在音频的回放以及音频的输入。对于音频回放效果，我们将着眼于 Sonia 库以及 Minim 库。音频输入将通过 Sonia 库来进行演示。

音频库可以在以下的网址中找到：

Sonia: <http://sonia.pitaru.com/>
 Minim: <http://code.compartmental.net/tools/minim/>

关于如何安装第三方库请回顾第12章。你同样也可以访问 <http://www.learningprocessing.com> 来下载例子中所使用的音频样本文件。

20.1 简单的音频播放

在我们来研究库之间，这里有一个简单的方式可以让 Processing 草稿不需要第三方库支持来播放音频(但是有很多限制)。这是通过使用我们在第16章中所学到的 Movie 类来完成。Movie 类被设计用于播放 QuickTime

电影，但是它同样可以用于播放一个 WAV 或者 AIFF 格式文件。WAV 是一种音频文件格式，它的全称是“Waveform audio format”(波形音频格式)这是 PC 中的标准格式。AIFF 全称是“Audio Interchange File Format”(音频交替文件格式)它通常被用于 Mac 电脑。

所有我们在第 16 章中所看到的函数都能够对音频文件使用。当然有区别，主要区别在于我们不能用 `image()` 函数来显示 Movie 对象，因为音频文件不存在图像。

例 20-1：带有 video 库的简单音频播放

```
import processing.video.*;
Movie movie;

void setup() {
    size(200, 200);
    movie = new Movie(this, "dingdong.wav");
}

void draw() {
    background(0);
    noLoop();
}

void mousePressed() {
    movie.stop();
    movie.play();
}
```

通过一个 WAV(或者AIFF)格式文件
来构造一个 Movie 对象。

每当鼠标按下时音频就会播放。函数 `stop()` 会
放在 `play()` 之前，是为了确保音频文件在开始
的时候 ~~九~~ 在播放。



练习 20-1：如果你有一个 Arduino 电路板，运用你已经做好的草稿来建立属于你自己的交互控制。(在你尝试你自己的创意之前，你应该确保在本章中所提供的简单的例子能够成功运行)

20.2 利用 Sonia 和 Minim

Movie 类能够做一些简单的淫笑播放，但是对于更复杂的回放功能，我们需要看看专门用于音频所设计的库。我们将从 Sonia 库开始 (<http://sonia.pitaru.com/>)。首先第一件事，对于任何第三方库，我们都需要在顶部写出导入语句。

```
import pitaru.sonia_v2_9.*;
```

Sonia 使一个连接 Java 音频库的接口。为了能够工作，Sonia 需要在音频播放和硬件输入之间建立一个通信。这个通信已经在 Java 音频库中完成。在任何音频被播放之前，有一个 link 需要被建立，那就是开始的音频声音。通过下面的代码进行完成，你需要放在 `setup()` 中。

```
void setup() {
    Sonia.start(this);
}
```

这样相当不负责的将声卡硬件连接了起来，当我们完成任务时却从没有想过要把它关闭。对于现在来说，Processing 草稿的流程都已经非常熟悉了，在 setup() 中开始，draw() 中循环，每当你退出草稿时就会停止。但是事情是，我们想在退出草稿时终止音频引擎。幸运的来说，我们这里有一个额外的隐藏函数能够让我们实现这个更能，stop()。当草稿退出，任何指令都能在 stop() 函数执行。在这里我们将关闭 Sonia 引擎。

```
void stop(){
    Sonia.stop();
    super.stop();
}
```

Sonia.stop() 被调用出来用于停止 Sonia。同样 super.stop()。“super”的含义我们会在第 22 章中进行进一步讨论。对于现在，我们能够理解这一行所说的意思：任何你想做的事通常都能在 stop() 中完成。

上面所讲的内容同样也都适用于 Minim。下面是相应的代码：

```
import ddf.minim.*;

void setup() {
    size(200, 200);
    Minim.start(this);
}

void stop() {
    super.stop();
}
```

Minim引擎连接开始

使用Minim，音频在这里可以单独
的被关闭。我们会在这里看到一个
时间。

20.3 基本的音频回放

一旦完成了利用 this 调用库的操作，你就拥有了整世界（声音的世界）。我们首先将演示如何播放音频文件。在音频被播放之前，它必须被加载到内存中，这和我们之前加载图像文件的方式相同。对于 Sonia 来说，Sample 对象用于存储声音样本。

```
Sample dingdong;
```

对象通过将音频文件名称传送到构造器中来初始化。

```
dingdong = new Sample("dingdong.wav");
```

就像图像一样，从硬盘中加载音频文件的过程缓慢，所以我们应该在 `setup()` 中提前加载，以便不妨碍 `draw()` 运行的速度。

兼容 Sonia 的文件类型还是比较有限的。只有 WAV 或者 AIFF (16-bit 单声道或立体声) 格式文件允许使用。如果你使用的文件不是兼容格式，你可能就需要下载一个免费的音频编辑器，如 Audacity (<http://audacity.sourceforge.net/>) 并转换格式。大多数音频文件会被保存为无损音质，或压缩音质。Sonia 只能在无损音质的文件下工作。但是在下一节，我们会演示如何使用 ESS 库播放压缩音质的格式文件 (MP3)。

```
dingdong.play();
一旦音频被加载，播放就很简单了：
```

下面的例子演示了每当鼠标点击圆形门铃的时候就会播放门铃的声音。门铃的类能够实现一些简单的按钮功能 (点击和侧翻)，并且这是对练习9–8一个很好的解决方案。下列代码中用粗体表示的是一些新的概念(音频播放相关)。

例 20-2：带有 Sonia 库的门铃

```
// Import the Sonia library
import pitaru.sonia_v2_9.*;

// A sample object (for a sound)
Sample dingdong;

// A doorbell object (that will trigger the sound)
Doorbell doorbell;

void setup() {
  size(200,200);
  Sonia.start(this); // Start Sonia engine.
  // Create a new sample object.
  dingdong = new Sample("dingdong.wav");
  // Create a new doorbell
  doorbell = new Doorbell(150,100,32);
  smooth();
}

void draw() {
  background(255);
  // Show the doorbell
  doorbell.display(mouseX,mouseY);
}
```

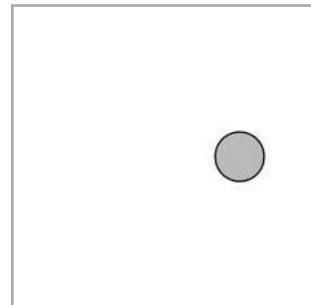


图 20.1

404

```
void mousePressed() {
    // If the user clicks on the doorbell, play the sound!
    if (doorbell.contains(mouseX,mouseY)) {
        dingdong.play();
    }
}

// Close the sound engine
public void stop() {
    Sonia.stop();
    super.stop();
}

// A Class to describe a "doorbell" (really a button)
class Doorbell {
    // Location and size
    float x;
    float y;
    float r;

    // Create the doorbell
    Doorbell (float x_, float y_, float r_) {
        x = x_;
        y = y_;
        r = r_;
    }

    // Is a point inside the doorbell (used for mouse rollover, etc.)
    boolean contains(float mx, float my) {
        if (dist(mx,my,x,y) < r) {
            return true;
        } else {
            return false;
        }
    }
}

// Show the doorbell (hardcoded colors, could be improved)
void display(float mx, float my) {
    if (contains(mx,my)) {
        fill(100);
    } else {
        fill(175);
    }
    stroke(0);
    ellipse(x,y,r,r);
}
```

练习20-2：重写门铃的类，包裹 Sonia 所引用的音频样本。让你拥有多个门铃对象，并且让每个门铃对象都有不同的声音。这里有一些代码给你开头。



```
// A Class to describe a "doorbell" (really a button)

class Doorbell {
    // Location and size
    float x;
    float y;
    float r;
    // A sample object (for a sound)
    _____ _____;

    // Create the doorbell
    Doorbell (float x_, float y_, float r_, _____ filename) {
        x = x_;
        y = y_;
        r = r_;
        _____ = new _____(_____);
    }

    void ring() {
        _____;
    }

    boolean contains(float mx, float my) {
        // same as original
    }

    void display(float mx, float my) {
        // same as original
    }
}
```

如果你运行门铃的例子，并且连续快速的点击门铃按钮，你会注意到每次点击的时候声音都会重新开始。它没有给你机会来完成播放。虽然对于这个简单的例子来说，没有太大的问题，但是让声音停止重新开始是很重要的，也会更复杂。最简单的方式就是在你调用 play() 函数之前检查音频是否在播放。Sonia 有一个很好的函数可以用来检查，那就是 isPlaying()，它会返回 true 或者 false。我如果现在没有音频在播放，那么我们就可以播放音频了，就像这样：

```
if (!dingdong.isPlaying()) {
    dingdong.play();
}
```

 请注意“!”意味着没有，不是

练习20-3：将门铃的类扩展成当音频在播放时出现动画(可能是移动它的位置以及随机的改变它的大小)。创建一组包含5个门铃对象的数组。这里是一部分用于添加到门铃类的函数：



```
void jiggle() {
    if (_____)) {
        x += _____;
        y += _____;
        _____;
    }
}
```

如果是使用 Minim 库来代 Sonia，变化非常小。代替之前这 Sample 对象，这里我们使用一个 AudioPlayer 对象。下面的代码包含了练习20-2和 20-3的解决方案。这里并没有用用数组来完成。

例 20-3：带有Minim库的门铃

```
import ddf.minim.*;

// A doorbell object (that will trigger the sound)
Doorbell doorbell;

void setup() {
    size(200,200);
    // start up Minim
Minim.start(this);
    // Create a new doorbell
    doorbell = new Doorbell(150,100,32, "dingdong.wav");
    smooth();
}
```

```

void draw() {
    background(100,100,126);
    // Show the doorbell
    doorbell.display(mouseX,mouseY);
    doorbell.jiggle();
}

void mousePressed() {
    // If the user clicks on the doorbell, play the sound!
    if (doorbell.contains(mouseX,mouseY)) {
        doorbell.ring();
    }
}

// Close the sound files
public void stop() {
    doorbell.close();
    super.stop();
}

class Doorbell {
    // Location and size
    float x;
    float y;
    float r;
    // An AudioPlayer object
    AudioPlayer dingdong;

    // Create the doorbell
    Doorbell (float x_, float y_, float r_, String filename) {
        x = x_;
        y = y_;
        r = r_;
        // load "dingdong.wav" into a new AudioPlayer
        dingdong = Minim.loadFile(filename);
    }

    // If the "doorbell" is ringing, the shape jiggles
    void jiggle() {
        if (dingdong.isPlaying()) {
            x += random(-1,1);
            y += random(-1,1);
            r = constrain(r + random(-2,2),10,100);
        }
    }
}

// The doorbell rings!
void ring() {
    if (!dingdong.isPlaying()) {
        dingdong.rewind();
        dingdong.play();
    }
}

```

门铃对象必须关闭它的声音

一个 AudioPlayer 对象用来存储声音

如果音频在播放，门铃只会摇晃

ring() 函数会播放音频，只要它没有在播放中。rewind() 会确保音频是从头开始的。

```

// Is a point inside the doorbell (used for mouse rollover, etc.)
boolean contains(float mx, float my) {
    if (dist(mx,my,x,y) < r) {
        return true;
    } else {
        return false;
    }
}

// Show the doorbell (hardcoded colors, could be improved)
void display(float mx, float my) {
    if (contains(mx,my)) {
        fill( 126,114,100);
    } else {
        fill(119,152,202);
    }
    stroke(202,175,142);
    ellipse(x,y,r,r);
}

void close() {
    dingdong.close();
}
}

```

门铃有一个 **close()** 函数用来关闭 AudioPlayer 对象

Minim 比 Sonia 库高级一点的是 Minim 支持 MP3 格式文件。MP3 (全称 “MPEG-1 Audio Layer 3”) 格式文件是压缩文件，因此会比无损的 WAC,AIFF 音频格式所占用的硬盘空间要小很多。

```
AudioPlayer dingdong = Minim.loadFile("dingdong.mp3");
```

20.4 有点发抖的声音回放

在音频回放时。我们能够通过 Sonia 和 Minim 库来实时操作音频样本的音量、音调以及声道。

让我们先使用 Sonia 库来控制音量。Sample 对象的音量可以通过 setVolume() 函数进行设置，这个函数需要一个 0.0 到 1.0 之间的浮点值作为参数 (0.0 是没有声音，1.0 是声音最大)。下面代码段假设一个音频样本名叫 “tone” ，我们将基于 mouseX 的坐标位置来设定它的音量值。(通过某种比例关系让它的值在 0 到 1 之间)。

```
float ratio = (float) mouseX /width;
tone.setVolume(ratio);
```

音量范围保持在 0.0 到 1.0 之间

声道的工作原理同样如此，只是他的范围是 -1.0 (左) 到 1.0 (右) 之间。

```
float ratio = (float) mouseX /width;
tone.setPan(ratio*2 - 1);
```

声道范围保持在 -1.0 到 1.0 之间

音调的控制通过改变回放的速率来调整(即加快播放速率声调会提高，降低播放速率声道会降低)，使用 **setRate()**。你可以设置任何一个你喜欢的音频样本进行速率的调整，它的范围在 0 (慢到你听不到) 到 88200 之间 (一个相对较快的速率)。

```
float ratio = (float) mouseX / width;
tone.setRate(ratio*88200);
```

一个合理范围内的速率，范围是0到
88200之间

下面的例子可以通过鼠标的移动来调整音频的音量以及音调。注意使用 repeat() 来代替 play()，它会一遍又一遍的重复循环，而不是只播放一遍。

例 20-4：控制音频(利用 Sonia 库)

```
// Import the Sonia library
import pitaru.sonia_v2_9.*;

// A Sample object (for a sound)
Sample tone;

void setup() {
    size(200,200);
    Sonia.start(this); // Start Sonia engine.
    // Create a new sample object.
    tone = new Sample("tone.wav");
    // Loop the sound forever
    // (well, at least until stop() is called)
    tone.repeat();
    smooth();
}

void draw() {
    if (tone.isPlaying()) {
        background(255);
    } else {
        background(100);
    }

    // Set the volume to a range between 0 and 1.0
    float ratio = (float) mouseX / width;
    tone.setVolume(ratio);

    // Set the rate to a range between 0 and 88,200
    // Changing the rate alters the pitch
    ratio = (float) mouseY / height;
    tone.setRate(ratio*88200);

    // Draw some rectangles to show what is going on
    stroke(0);
    fill(175);
    rect(0,160	mouseX,20);
    stroke(0);
    fill(175);
    rect(160,0,20	mouseY);
}
```

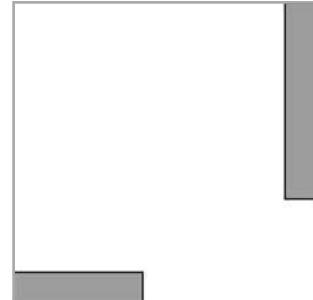


图 20.2

通过 mouseX 的位置来设
定音量

通过 mouseY 的位置来设
定音速率

```
// Pressing the mouse stops and starts the sound
void mousePressed() {
    if (tone.isPlaying()) {
        tone.stop();
    } else {
        tone.repeat();
    }
}

// Close the sound engine
public void stop() {
    Sonia.stop();
    super.stop();
}
```

使用 **stop()** 函数能够让音频停止



练习20-4：让例20-4通过翻转Y坐标轴来控制声音，而不是通过鼠标的上升或者下降。

在使用Minim库的情况下，我们同样也可以使用下列函数来控制 `AudioPlayer()` 对象：`setVolume()`, `setPan()`, `addEffect()`，以及一些其他的控制微量(详细可以访问：<http://code.compartmental.net/tools/minim/>)。

20.5 实时输入

在第16章中，我们研究了如何通过串口通信允许从一个外部硬件设备连接到传感器在 Processing 草稿中进行输入的响应。从麦克风输入也是同样如此。在本质上来说，麦克风就是一个传感器。麦克风不仅仅可以纪录声音，还能判断声音的大小，音调等等。例如，Processing 草稿可以基于声音的登记来判断这个声音是否在一个拥挤的房间内产生的，或者通过音调来判断女高音或者男低音。这一节我们将学习如何利用 `Sonia` 库检索以及使用声音的音量数据。关于分析麦克风声音音调的等级的相似内容请访问 `Sonia` 网站 (<http://sonia.pitaru.com>)去进一步了解。

在之前的章节我们使用 `Sample` 对象来播放音频。现在我们需要使用 `LiveInput` 对象来对麦克风声音的输入进行检索。这里我们会使用我们到现在为止在这本书中还没有遇到过的 2 个类。

思考这样一个场景，我们又 3 个音频文件。我们要创建 3 个 `Sample` 对象。

```
Sample sample1 = new Sample("file1.wav");
Sample sample2 = new Sample("file2.wav");
Sample sample3 = new Sample("file3.wav");
```

从技术上讲，我们已经做好了 `Sample` 对象的 3 个实例，通过 `Sample` 类来完成。如果我们想播放一个音频，我们必须制定一个 `Sample` 对象。

```
sample1.play();
```

但是对于 LiveInput 来说我们讲不会有任何的对象实例。因为 Sonia 库只允许 1 个输入源，所以这里没有理由去指定一个 LiveInput 对象。相反，当在听麦克风的时候，我们会将 LiveInput 类作为一个整体：

```
LiveInput.start(); // Start LiveInput
```

调用的函数会跟在类的名称后面(而不是任何指定的对象实例名称后面)，我们将这些函数称为静态函数。我们不能在 Processing 中创建自己的静态函数，所以我们对这个不需要了解的太深。但是他们是 Java 的一部分，我们在使用第三方库时有时候会碰到这些静态函数。liveInput 时一个检测麦克风声音以及音调等级的带有静态函数的类的例子。

让我们开始建立一个非常简单的例子，将圆的尺寸与音量的等级进行联系

第一步，明确我们要做的，开始接听电脑麦克风这个过程。

```
LiveInput.start(); // Start LiveInput
```

第二步，从麦克风中读取音量等级。我们可以用2种方式来操作。因为麦克风是通过立体声进行监听，哦我们可以通过 **getLevel()** 来读取左声道或者右声道的音量等级。

```
float rightLevel = LiveInput.getLevel(Sonia.RIGHT);
float leftLevel = LiveInput.getLevel(Sonia.LEFT);
```

我们同样也可以用简单的方式监听麦克风音量等级

```
float level = LiveInput.getLevel();
```

音量等级始终会保持在0.0到1.0之间。

我们可以指定我们想要监听左声道还是右声道

如果里面不带参数，**getLevel()** 会返回左声道和右声道的综合音量

将它们放在一起，利用音量大小的等级来绘制圆圈的尺寸，我们就有：

例 20-5：带有 Sonia 库的实时输入

```
// Import the Sonia library
import pitaru.sonia_v2_9.*;

void setup() {
    size(200,200);
    Sonia.start(this);
    // Start listening to the microphone
    LiveInput.start();
    smooth();
}
```

所有音频输入的函数都是静态的，这意味着我们可以直接在类的名称后面书写函数，LiveInput，而不是一些其他的什么对象实例。

Sonia 库的实时输入会监听你电脑种所设定的“音频输入”设备的所有声音。你可以使用一个内置麦克风，可以从另一个声源进行连接，甚至是在你电脑种的 CD 播放器。Sample 对象同样也可以被当作实时输入，这要通过 **connectLiveInput()** 来实现。

```

void draw() {
    background(255,120,0);
    // Get the overall volume (between 0 and 1.0)
    float level = LiveInput.getLevel();
    fill(200);
    stroke(50);
    // Draw an ellipse with size based on volume
    ellipse(width/2,height/2,level*200,level*200);
}

// Close the sound engine
public void stop() {
    Sonia.stop();
    super.stop();
}

```

变量能够存储音量 (“level”),
并用于圆圈的尺寸



练习20-5：重写例 20-5,让实时输入的左声道和右声道音量分别映射在不同的圆圈尺寸上。

20.5 音频的阈值

当音频产生时，一种常见的声音交互就是触发事件。思考“拍子”。拍手时，灯就会亮起来，再拍一次灯就会暗下去。牌子可以被理解成一个非常响亮和简短的声音。要在 Processing 对“拍子”进行编程，我们需要监听音量，并且要让音量高的时候触发一个事件。

在用户拍手的情况下，我们可能会觉得它的音量等级应该大于 0.5 (这不是一个标准的科学测量，但是对于这个例子来说足够了)。0.5 这个值被称之为阈值。如果高于这个阈值，事件就会触发；如果小于这个阈值，事件就不会被触发。

```

float vol = LiveInput.getLevel();
if (vol > 0.5) {
    // DO SOMETHING WHEN THE VOLUME IS GREATER THAN ONE!
}

```

例 20-6 中，每当音量等级大于 0.5 时，窗口中就会绘制矩形。音量等级同样会显示在左侧栏中。

例 20-6：带有 Sonia 库的阈值

```

// Import the Sonia library
import pitaru.sonia_v2_9.*;

void setup() {
    size(200,200);
    Sonia.start(this); // Start Sonia engine.
    LiveInput.start(); // Start listening to the microphone
    smooth();
    background(255);
}

```

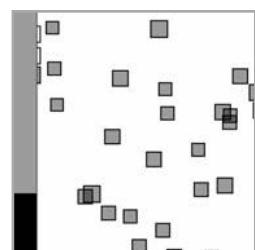


图 20.3

```

void draw() {
    // Get the overall volume (between 0 and 1.0)
    float vol = LiveInput.getLevel();
    // If the volume is greater than 0.5, draw a rectangle

    if (vol > 0.5) {
        stroke(0);
        fill(0,100);
        rect(random(width),random(height),vol*20,vol*20);
    }

    // Graph the overall volume
    // First draw a background strip
    fill(175);
    rect(0,0,20,height);
    // Then draw a rectangle size according to volume
    fill(0);
    rect(0,height-vol*height/2,20,vol*height/2);
}

// Close the sound engine
public void stop() {
    Sonia.stop();
    super.stop();
}

```

如果音量等级大于0.5，窗口中就会在随机的坐标中出现矩形。音量越大，矩形的尺寸就越大。

这个应用工作的还算不错，但没有真正的模拟拍手的声音。请注意每次拍手的结果就是在窗口中会出现几个矩形。这时因为声音的发生持续了一段时间。它可能持续了非常短的一段时间，但是这时间足够让大于0.5音量等级的音量在draw()循环好几次。

为了让每次拍手事件只出现一次画面，我们需要重新思考我们的编程逻辑。伪代码，我们正在努力实现：

- 如果音量等级大于0.5，那么你是在拍手，并且会触发事件。但是，如果你只是刚刚才做了拍手，事件就不会触发。

这里的关键是我们怎么去明确“刚刚”这个含义。一种方式是采取定时器，也就是只要触发事件，那么它下一次触发事件必须是在1秒之后发生。这是一个完美的解决方案。尽管如此，对于声音来说定时器是一个完全没必要去做的工作，因为声音的音量本身就会告诉我们一次拍手什么时候完成。

- 如果声音等级小于0.25，那么它会很安静，我们就完成了拍手这个动作

OK,根据这两条逻辑，我们可以准备一个“双阈值”算法编程。我们可以使用2个阈值，一个用于确定我们是否开始了拍手，另一个用来确定我们是否完成了拍手这个动作。我们需要一个boolean变量来告诉我们现在是否有拍手。

假设 拍手 = 假启动

- 如果音量等级在0.5以上并且我们已经没有拍手，事件触发，设置拍手为true。
- 如果我们拍手并且音量等级小于0.25，那么就是安静的，设置拍手为false。

转换成代码，就是如下：

```
// If the volume is greater than one and we are not clapping, draw a rectangle
if (vol > 0.5 && !clapping) {
    // Trigger event!
    clapping = true; // We are now clapping!
} else if (clapping && vol < 0.25) { // If we are finished clapping
    clapping = false;
}
```

这里是完整的例子，每次拍手有且仅有一个矩形显示。

例 20-6：带有 Sonia 库的阈值

```
// Import the Sonia library
import pitaru.sonia_v2_9.*;

float clapLevel = 0.5; // How loud is a clap
float threshold = 0.25; // How quiet is silence
boolean clapping = false;

void setup() {
    size(200,200);
    Sonia.start(this); // Start Sonia engine.
    LiveInput.start(); // Start listening to the microphone
    smooth();
    background(255);
}

void draw() {
    // Get the overall volume (between 0 and 2.0)
    float vol = LiveInput.getLevel();
    // If the volume is greater than 0.5 and
    // we are not clapping, draw a rectangle
    if (vol > clapLevel && !clapping) {
        stroke(0);
        fill(0,100);
        rect(random(width),random(height),vol*20,vol*20);
        clapping = true; // We are now clapping!
        // If we are finished clapping
    } else if (clapping && vol < threshold) {
        clapping = false;
    }

    // Graph the overall volume
    // First draw a background strip
    noStroke();
    fill(200);
    rect(0,0,20,height);
    // Then draw a rectangle size according to volume
    fill(100);
    rect(0,height-vol*height/2,20,vol*height/2);
    // Draw lines at the threshold levels
    stroke(0);
    line(0,height-clapLevel*height/2,19,height-clapLevel*height/2 );
    line(0,height-threshold*height/2,19,height-threshold*height/2 );
}
```

```
// Close the sound engine
public void stop() {
    Sonia.stop();
    super.stop();
}
```

练习20-6：让例20-7中的触发事件编程2次拍手才会触发。这里是一些代码帮助你开头，假设一个存在的变量“clapCount”。



```
if (vol > c_lapLevel && !clapping) {
    clapCount____;
    if (_____) {
        _____;
        _____;
        _____;
    }
    _____;
} else if (clapping && vol < 0.5) {
    clapping = false;
}
```



练习20-7：创建一个声控的简单游戏。建议：最好先利用鼠标来控制游戏，然后将实时输入的音量等级替换掉鼠标坐标。有些例子是乒乓球，球拍的位置取决于声音音量的大小。或者是打猎，当用户拍手时，子弹会发射。

21. 导出

“等一下。我想艺术品要放弃被导出。”

—Seinfeld

本章内容：

- 网络小程序
- 独立应用
- 高分辨率 PDF
- 图像和图像序列
- QuickTime 视频

在这本书中我们集中精神来学习编程。但是最终我们的代码婴儿要长大，并且想在世界上找到属于它们的路。本章时专门讨论 Processing 草稿可以发布导出的各种选项。

21.1 网络小程序

Processing 草稿可以导出为 Java 小程序在 web 上运行。关于怎么工作的，我们在第二章已经讲过。然而我们已经看见因为各种安全限制，小应用不能执行某些任务，如连接远程服务器或者访问串口，网络摄像头或者是其他硬件设备。如果你有一个草稿必须发布在网上，会有一些限制，但也有一些解决方案。一种可能(第16章中提到的)是给予你的应用一个签署协议。要做到这一点你可以从本书的网站中获得更多提示 (<http://www.learningprocessing.com/sandbox>)。

21.2 独立应用

Processing 有一个很大的特点，那就它能够将草稿直接导出成一个独立的应用，这意味着你只需要双击这个应用并且不需要任何 Processing 开发环境就能够运行。

如果你正在为安装环境创建一个编程，它也会允许你在电脑开启后轻松的创建应用。如果你要在同一事件运行一个草稿的多个副本，导出应用来运行也是非常有用的(如在第 19 章中客户端与服务器的例子)。在应用中没有类似小程序那样的安全限制。

导出应用应该：

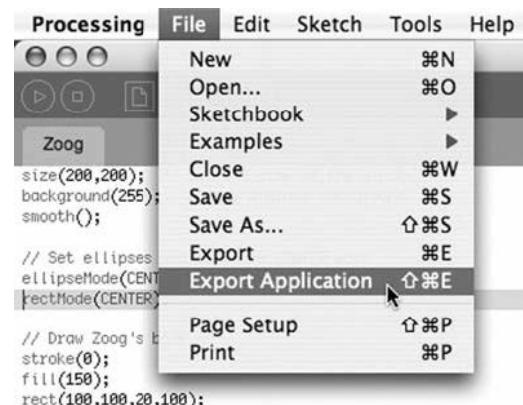


图 21.1

FILE → EXPORT APPLICATION (如图21.1所示)

然后你会注意到出现了3个新的文件夹，如图21.2所示。



图 21.2

Processing会为3个操作系统自动创建相应的应用，Mac OS X, Windows, 和 Linux。如果你是从Windows电脑中导出应用，Mac应用就无法正常工作(关于修复此问题的说明都包含在readme.txt文件中)。为了避免这些，我们只需要在Mac上导出应用。

这个文件夹包含所有你所需要的文件：

- sketchName.exe (在Mac和Linux系统上也被称为sketchName)。此文件是双击即可运行的应用文件。
- “source”目录。应用文件夹中会有这个目录，这个目录包含了所有你编程的源文件。你运行程序时不需要这些文件。
- “lib”目录。这个文件夹只显示在Windows和Linux系统里面，它会包含所需要的库的文件。它通常会包含core.jar Processing的核心库，同样也有你导入的一些库。在Mac OS X里面，如果你要看见这些内容，就通过双击应用文件，并且选择“Show Package Contents”。

运行应用的用户需要确保他的电脑上已经安装了Java。在Mac中，Java时预装在操作系统中的，所以只要你不乱动你的Java文件就不会出现任何问题。在Windows中，会涉及到这种情况，你可以通过复制Processing目录中的Java文件夹到应用文件夹中来包含Java(你必须在PC上做这一点)。

这里有一些关于导出应用的一些小技巧。例如，如果你想在标题栏中有你自己想要的文字，你可以添加下列代码到setup()中。

```
frame.setTitle("My super-awesome application!");
```

此外，虽然导出的应用程序通常在窗口模式下运行，你也可以通过添加一些代码来让它全屏运行。理解这些代码需要知道一些高级的Java知识，以及Processing草稿的内部结构。

一些相关的线索会在第23章中透露，但是现在，你只需要复制下面提供的代码：

```
static public void main(String args[]) {
    PApplet.main(new String[] { "-present", "SketchName" });
}
```

这个代码可以插入编程中的任何位置(通常是放在 `setup()` 上面)。” “SketchName” 必须与你 Processing 草稿的名称匹配。



练习21-1：以独立应用的方式导出一个你已经完成的Processing草稿。

21.3 高分辨率PDF

我们一直在使用 Processing，并将其作为我们创建平面图形的一种手段。实际是，如果回归第3章，我们花了大量的时间去学习编程工作的流程。不过，现在是一个很好的机会来让想法回归到一个静态编程中，其目的就是创建一个静态图像。Processing 的 PDF 库允许我们采用这些静态草稿，并且创建出高分辨率的图像用于打印。(实际上，这本书中几乎所有的图像都是通过这种方式产生的)

要使用 PDF 库，下面的步骤是必须的：

第一步，导入 PDF 库

```
import processing.pdf.*;
```

第二步，在 `setup()` 中使用 `size()` 函数，并添加为 PDF 模式，以及一个字符串的文件名。

```
size(400, 400, PDF, "filename.pdf");
```

第三步，在 `draw()`, 做你想做的任何事

```
background(255);
fill(175);
stroke(0);
ellipse(width/2,height/2,160,160);
```

第四步，调用 `exit()` 函数，这个非常重要。`exit()` 函数用来完成 PDF 的渲染。如果没有调用，这个文件将无法正常打开。

```
exit(); // Required!
```

下面是把上面的步骤放到一起的一个例子。

例 21-1：基础 PDF 操作

```
// Import the library
import processing.pdf.*;

// Using "PDF" mode, 4th argument is the name of the file
size(400, 400, PDF, "filename.pdf");

// Draw some stuff!
background(255);
fill(175);
stroke(0);
ellipse(width/2,height/2,160,160);

// Very important, required for the PDF to render properly
exit();
```

如果你运行这个例子，你会注意到没有窗口显示出来。一旦你设置了 Processing 的渲染模式为 PDF，草稿窗口就不会再出现。这时因为 PDF 模式通常会创造极高的分辨率，复杂的图像不能再屏幕上显示的很好。

但是，当你再渲染 PDF 的时候使用函数 beginRecord() 和 endRecord()，你就能够再窗口中显示运行的草稿。运行的速度会比第一个例子慢，但是它能够让你看到保存的时候是什么样子的。

例 21-2：使用 beginRecord() 来操作 PDF

```
import processing.pdf.*;
```

```
void setup() {
    size(400, 400);
    beginRecord(PDF, "filename.pdf");
}
```

beginRecord() 开始这个过程。第一个参数应该是读取 PDF 模式，第二个参数应该是 PDF 的文件名称。

```
void draw() {
    // Draw some stuff!
    smooth();
    background(100);
    fill(0);
    stroke(255);
    ellipse(width/2,height/2,160,160);
    endRecord();
}
```

调用 **endRecord()** 函数来完成 PDF

```
}  
noLoop();
```

因为 PDF 已经完成，所以我们不需要再循环

endRecord() 不需要再第一帧渲染的时候就一定要调用，因此我们可以在 draw() 中多一些循环周期之后在生成 PDF。下面的例子使用了第 16 章 “Scribbler”的例子，并且渲染成一个 PDF。这里颜色的来源不是来自于视频，而是来自于计数器变量。图 21.3 是输出样本。

例 21-3：多帧放入一个PDF

```

import processing.pdf.*;

float x 0;
float y 0;

void setup() {
    size(400, 400);
    beginRecord(PDF, "scribbler.pdf");
    background(255);
}

void draw() {
    // Pick a new x and y
    float newx = constrain(x + random( -20,20),0,width);
    float newy = constrain(y + random( -20,20),0,height);
    // Draw a line from x,y to the newx,newy
    stroke(frameCount % 255,frameCount*3 % 255,
    frameCount*11 % 255,100);
    strokeWeight(4);
    line(x,y,newx,newy);
    // Save newx, newy in x,y
    x newx;
    y newy;
}

// When the mouse is pressed, we finish the PDF
void mousePressed() {
    endRecord();
    // We can tell Processing to open the PDF
    open(sketchPath( "scribbler.pdf"));
    noLoop();
}

```

background() 应该放置在 **setup()** 中。如果 **background()** 放置在 **draw()** 中，PDF 会累计大量的图形元素，并且是反复被删除的图形元素。



图 21.3

在这个例子中，用户会通过双击鼠标来完成 PDF 的渲染。

如果是渲染 3D 形状(在 P3D 或 OPENGL 模式下)，你会需要使用 **beginRaw()** 和 **endRaw()** 来代替 **beginRecord()** 和 **endRecord()**。这个例子同样也使用了一个 **boolean** 变量(“ recordPDF ”)。

例 21-2：使用 **beginRecord()** 来操作 PDF

```

// Using OPENGL
import processing.opengl.*;
import processing.pdf.*;

// Cube rotation
float yTheta = 0.0;
float xTheta = 0.0;

// To trigger recording the PDF
boolean recordPDF = false;

void setup() {
    size(400, 400, OPENGL);
    smooth();
}

```

当 **boolean** 变量的值为 **true** 时，PDF 就会被做出来。

在 OPENGL 或 P3D 模式下，**beginRaw()** 和 **endRaw()** 会代替 **beginRecord()** 和 **endRecord()**

```

void draw() {
    // Begin making the PDF
    if (recordPDF) {
        beginRaw(PDF, "3D.pdf");
    }

    background(255);
    stroke(0);
    noFill();
    translate(width/2,height/2);
    rotateX(xTheta);
    rotateY(yTheta);
    box(100);
    xTheta += 0.02;
    yTheta += 0.03;
    // End making the PDF
    if (recordPDF) {
        endRaw();
        recordPDF = false;
    }
}

// Make the PDF when the mouse is pressed
void mousePressed() {
    recordPDF = true;
}

```

如果在文件名中包含“####”，如“3D-####.pdf”，这代表着一系列PDF都是在每一帧的时候进行单独渲染的。

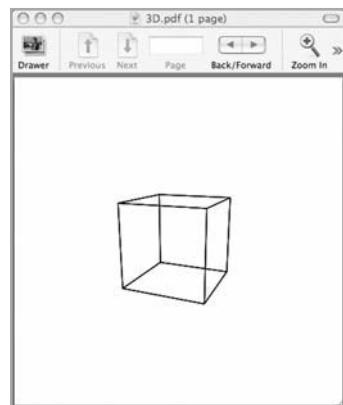


图 21.4

这里有关于 PDF 库的 2 个重点需要注意：

- Images—如果在 PDF 显示图像，也不会比原导入文件的效果要好。例如 320*240 像素的图像仍然还是一个 320*240 像素的图像，不管你是不是导出为一个高分辨率的 PDF。
- Text—如果你在 PDF 中显示文本，你必须安装字体才能正确的在其他没有安装你渲染文本的字体的电脑上看 PDF。有一种解决方法是在 size() 后面包含 “textMode(SHAPE);”。这将把文本字轮廓方渲染在 PDF 中，所以其他电脑上就不需要安装相应字体去查看了。

关于 PDF 库的完整档案，请查看 Processing reference 页面 (<http://processing.org/reference/libraries/pdf/index.html>) 虽然 PDF 库可以满足你对高分辨率的需求，但是这里还有其他的 2 个库你可能感兴趣。一个是 proSVG 库，它能将图形导出为 SVG (“Scalable Vector Graphics”/可缩放矢量图形) 格式：<http://www.texone.org/prosvg/>。另一个是 SimplePostScript 库，它能将适量文件写入 PostScript 格式中：<http://processing.unlekker.net/SimplePostscript/>。



练习21-2：从任何 Processing 草稿中，创建一个 PDF。

21.4 Images/saveFrame()

高分辨率的 PDF 可以用于印刷；但是你可能还会想在 Processing 窗口中存储一些图像文件(只需要窗口像素大小的文件)。这个需要通过 save() 或者 saveFrame() 来完成。

`save()` 函数需要一个参数，那就是你想要存储的图片的名称。`save()` 可以生成下列格式的文件：JPG,TIF,TGA 或 PNG，这些都是文件的扩展名。如果没有这些扩展名包括在内，Processing 会默认导出为TIF格式。

```
background(255,0,0);
save("file.jpg");
```

如果你多次调用 `save()`，并且名称相同，那么之前所生成的图片都将被覆盖。但是你想存储一系列有序的图片，`saveFrame()` 函数能够按照自动编号来存储文件。Processing 会查找字符串“####”，并且将有编号的图像序列排出。如图 21.5 所示。

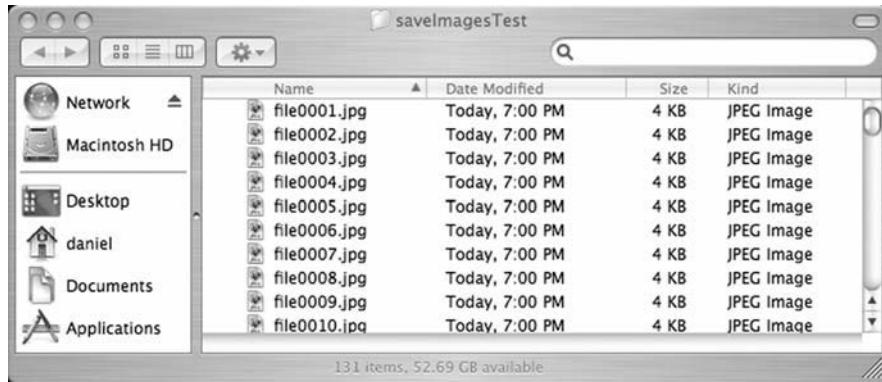


图 21.5

```
void draw() {
    background(random(255));
    saveFrame("file####.jpg");
}
```

这种技术通常用于将一系列有序编号图片导入视频或动画软件中。

21.5 视频制作

从 `saveFrame()` 中导出的序列图片能够使用 QuickTime Pro, iMovie, 或者任何视频打包软件制作成视频文件。

然而对于较长的视频，一个目录中出现成千上万个图像确实有些累赘。使用 MovieMaker 类能够通过将这些帧直接写入视频文件中让过程更简单。我最初开发 MovieMaker 将其作为第三方库，但是现在它已经存在于 Processing 中 video 库中了。

```
import processing.video.*;
```

一个 MovieMaker 对象需要下列参数(按顺序来)：

- **this**—参考视频相关的草稿
- **width**—整数，通常和草稿的宽度一样

- **height**—整数，通常和草稿的高度一样
- **filename**—字符串，作为视频的名称
- **framerate**—视频的帧速率，通常为**30**
- **type**—视频使用的压缩格式
- **quality**—视频压缩的质量

一个函数构造器的例子如下：

```
mm = new MovieMaker(this, width, height, "test.mov", 30, MovieMaker.H263, MovieMaker.HIGH);
```

这个库提供了许多编解码器的选项。术语编码最初来自于“coder-decoder/编解码器”，指的是将视频信号转换成数字信号的硬件设备。但是这里编码指的是“compressor-decompressor/编解码器”。软件能够在源有无压缩视频(用于观看)以及压缩视频(用于存储)之间进行转换。下面是 MovieMaker 库中能用的编解码器(参考随时有更新，

```
ANIMATION, BASE, BMP, CINEPAK, COMPONENT, CMYK, GIF, GRAPHICS, JPEG, MS_VIDEO, MOTION_
JPEG_A, MOTION_JPEG_B, RAW, SORENSEN, VIDEO, H261, H263, H264
```

请访问：<http://processing.org/reference/libraries/video/MovieMaker.html>

编解码器的选择会影响视频的尺寸，同样也会影响视频的质量。例如 RAW 会储存未被压缩的视频(因此质量最高)，但是会导致视频文件占用空间巨大。关于各种编解码器的更多相关内容，你可以到苹果的 QuickTime 网站去了解。

- QuickTime: <http://www.apple.com/quicktime/>.
- QuickTime Developer: <http://developer.apple.com/quicktime/>.

一旦你选择了一个编解码器，您必须指定的编解码器的质量，范围从最差到无损。

```
WORST, LOW, MEDIUM, HIGH, BEST, LOSSLESS
```

在完成 MovieMaker 函数构造器之后，我们可以通过调用 addFrame() 函数来在某一时刻添加帧。

```
void draw() {
    // Some cool stuff that you are drawing!
    mm.addFrame();
}
```

最后，为了能够让视频能够正常播放，我们必须使用 finishMovie() 函数来“完成”视频。如果你在这个函数还没有被调用的情况下退出 Processing 草稿，视频文件就不会被 QuickTime 认可。一种解决方式是通过鼠标点击来完成视频。但是如果在你的应用中已经使用了鼠标点击去做其他的什么操作，你就可能要想想其他的方法了，如按下一个什么键之后来完成视频之类的。

```
public void mousePressed() {
    mm.finishMovie();
}
```

例21-5纪录了鼠标在屏幕上画画的视频。在按下空格键时，视频会完成。



图 21.6 例21-5的输出图

例 21-5：制作一个 QuickTime 视频

```
import processing.video.*;
MovieMaker mm; // Declare MovieMaker object

void setup() {
    size(320, 240);
    // Create MovieMaker object with size, filename,
    // framerate, compression codec and quality
    mm = new MovieMaker(this, width, height, "drawing.mov", 30, MovieMaker.H263,
    MovieMaker.HIGH);
    background(255);
}

void draw() {
    stroke(0);
    strokeWeight(4);
    if (mousePressed) {
        line(pmouseX, pmouseY, mouseX, mouseY);
    }
    mm.addFrame(); // Add window's pixels to movie
}

void keyPressed() {
    if (key == ' ') {
        println("finishing movie ");
        mm.finish(); // Finish the movie if space bar is pressed!
    }
}
```

MovieMaker 类是 Processing 的
video 库中的一部分。

在 draw() 中循环的每一个周期都会
被作为一个新的帧添加到视频中。

不要忘记完成视频！否则视频会无法正常播放。



练习 21-3：从任何 Processing 草稿中，创建一个视频。



第九课项目

1. 通过添加音频文件或者实时输入，将音频加入到 Processing 草稿中。
2. 使用 Processing 生成一些可以实时导出的图形。导出一个 PDF，再导出一个视频文件。

使用下面所提供的空间来设计你的草稿。

第十课

超越 Processing 的内容

22. 高级面向对象编程

23. Java

22. 高级面向对象编程

“你有没有你从来都没有想过的事？”

—Henry Drummond, *Inherit the Wind*

本章内容：

- 封装
- 继承
- 多态性
- 过载

在第8章中，我们介绍了面向对象编程 (“OOP”)。里面的主导原则是将数据和函数配对到一个类中。类事一个模板，并且从模板中我们可以生成对象实例，并且将他们存储在变量和数组中。虽然我们学习了如何编写类以及生成对象，但是我们并没有深层次的研究 OPP 的核心原则和高级功能。现在已经接近书的尾声(下一张我们将进入 Java 的世界)，是时候好好的回顾一下之前的，以及学习更高级的内容了。

在 Processing 和 Java 中的面向对象有明确的 3 个记本概念：封装，继承，和多态性。现在我们已经熟悉了它的封装概念；我们只是没有正式化对概念的理解以及术语的使用。继承和多态性是完全新的概念，我们将在本章中描述。(在本章结尾，我们也会快速的学习一些过载，它能够允许对象用多种方式调用函数构造器)

22.1 封装

要了解封装，让我们一起回顾 Car 类的例子来。让我们来到例子外面的世界，想想现实生活中的汽车对象，通过实际的驾驶来操作它：你，这是一个不错的夏天，你厌倦了编程并且选择在周末的时候前往海滩。在交通畅通的情况下，你会哼着歌，轻松的按着刹车和油门。

你驾驶的这个车就是被封装的。所以你开车就需要操作一些函数：steer(), gas(), brake(), and radio()。你知不知道引擎盖下面是什么？了不了解催化转换器如何连接到发动机或发动机如何连接到冷却器？阀门以及电线，齿轮是用来做什么的？当然，如果你是一个汽车修理工你可能对这个问题了如指掌。但是你如果不是，你也不必为了开车而学这些内部东西，这就是封装。

封装被定义为用户对象所隐藏的内部工作。

对于面向对象编程而言，对象的“内部运作”是数据(对象的一些变量)以及函数。“用户”的对象是你，程序员，能够生成对象实例并且在你的代码中使用他们。

现在提起来有什么好处？在第8章(以及这本书中所有的面向对象编程的例子)中我们强调了模块化和可重复使用的原则。这意味着，如果你已经想到了如何编程一个汽车，你就不需要一次一次麻烦的重新编代码来编写汽车，而是直接先组织好Car类，需要的时候就可以使用。

封装已经更进一步。面向对象编程不仅仅只是帮你组织你的代码。它也可以保护你犯错。如果你在编写代码的时候不乱用代码，就不太会破坏汽车。当然，有时候一辆车坏了，你需要修它，这时候你就需要打开引擎盖并且查看类本身的问题。

看看下面的例子。比如说你正在些一个BankAccount类(银行帐户)，将一个浮点变量作为你的银行帐户余额。

```
BankAccount account = new BankAccount(1000);
```

一个起始余额为 \$1000 的银行帐户对象。

你需要封装余额并将它隐藏。为什么？比方说你要从该帐户取钱，所以你需要从这个帐户中减去\$100。

```
account.balance = account.balance - 100.
```

直接访问余额变量取走 \$100

但是如果有一个取钱手续费呢？比如说\$1.25，我们应该怎么做，如果在这样以详细细节来编程我们在银行编程的工作迟早会丢掉。如果运用封装，我们就绝对可以留住这个工作。代码如下。

```
account.withdraw(100);
```

通过调用函数取走 \$100

如果withdraw()函数书写正确，我们永远就不会忘记手续费，因为每次取钱，这个函数都会被调用。

```
void withdraw(float amount) {
    float fee = 1.25;
    account -= (amount + fee);
}
```

这个函数能够确保钱被取走的同时也会扣除相应的手续费

这样做的另一个好处就是如果银行决定将手续费提高到\$1.50，它就能在BankAccount类中简单的修改手续费变量，并且所有的其他内容也能正常工作。

从技术上将，为了遵循封装原则，在类中的变量不能被直接访问，并且只能通过一个函数来检索出来。这就是为什么你会经常看见程序员使用了大量被称为getters和setters的函数，这些函数能够检索或者改变变量的值。这里是一个Point类(包含2个变量x和y，并且需要通过getters和setters来访问)的例子。

```

class Point {
    float x,y;

    Point(float tempX, float tempY) {
        x = tempX;
        y = tempY;
    }

    // Getters
    float getX() {
        return x;
    }

    float getY() {
        return y;
    }

    // Setters
    float setX(float val) {
        x = val;
    }

    float setY(float val) {
        if (val > height) val = height;
        y = val;
    }
}

```

变量 x 和 y 需要通过 **getter** 和 **setter** 函数来进行访问

getters 和 **setters** 能让我们保护变量的值。例如，在这里 y 的值永远不会大于草图的高度。

如果我们想生成点，并且递增 y 的值，我们可能需要这样做：

```
p.setY(p.getY() + 1);
```

代替了：
p.y = p.y + 1;

Java 实际上允许我们将变量作为“隐私”，从而来避免非法访问的(换句话来说，如果你尝试去访问，编程将不能运行)

```

class Point {
    private float x;
    private float y;
}

```

在 Processing 的例子中这是非常罕见的，变量可以设置成隐私，这意味着只能在类的内部中才能访问。但是在默认情况下变量和函数在 Processing 都是“public/公共的”。

虽然封装是面向对象编程的核心部分，并且当它在进行大型团队项目开发的时候非常重要，但是保持法律(就像上面 Point 对象中 y 的值)通常会很不方便，并且当碰见简单的 Processing 草稿的时候这就非常傻。如果你创建一个类并且直接可以直接访问 x, y 变量是最好的。在这本书中我们已经有好几次的例子都这么做了。

了解封装的原则是对你如何设计你的对象以及管理你的代码的一个很好的驱动力。在任何外部暴露你类本身的工作的时候你应该问问自己：这是必要的吗？这个代码应该放在函数中还是应该放在类中？最后，你会成为一个快乐的程序员，并且还能继续呆在银行工作。

22.2 继承

继承，是我们关于面向对象编程 3 个概念中的第二个，它能够允许我们基于现有的类的基础上创建新的类。

让我们来看看动物们的世界：狗，猫，猴子，熊猫，袋熊等等。让我们从狗类编程开始。一个狗的对象会需要一个年龄的变量(整数)，同样需要 eat(), sleep(), and bark() 函数。

```
class Dog {
    int age;

    Dog() {
        age = 0;
    }

    void eat() {
        // eating code goes here
    }

    void sleep() {
        // sleeping code goes here
    }

    void bark() {
        println("WOOF!");
    }
}
```

完成了狗的类，现在让我们来看看猫的类。

```
class Cat {
    int age;
    Cat() {
        age = 0;
    }

    void eat() {
        // eating code goes here
    }

    void sleep() {
        // sleeping code goes here
    }

    void meow() {
        println("MEOW!");
    }
}
```

 注意狗的类和猫的类拥有同样的变量 (age) 以及函数 (eat,sleep)。但是他们同样也有不同的函数 bark 和 meow。

可悲的是，当我们再去写鱼、马、考拉熊等等的动物的时候，这个过程会相当的乏味，我们会重写一遍又一遍的代码。但是相反，如果我们可以开发一个通用的动物类来描述动物的类型会更好。毕竟动物都需要 eat 和 sleep。所以我们就可以说以下内容。

- 狗是动物，并拥有动物的所有属性，可以做所有动物能做的事。但是此外还能 bark/吠。
- 猫是动物，并拥有动物的所有属性，可以做所有动物能做的事。但是此外还能 meow/喵。

继承就允许我们再编程中这么做。类可以从其他的类中继承属性(变量)以及行为(函数)。再这里我们就称狗类子类，动物类为父类。子类会自动继承其父类的所有变量以及函数。子类同样也包含了父类中所没有的变量以及函数。继承遵循下面的树状结构(很像一个进化的“生命之树”)。狗类可以继承犬类，犬类可以继承哺乳动物类，哺乳动物类可以继承动物类等等。如图 22.1 所示。

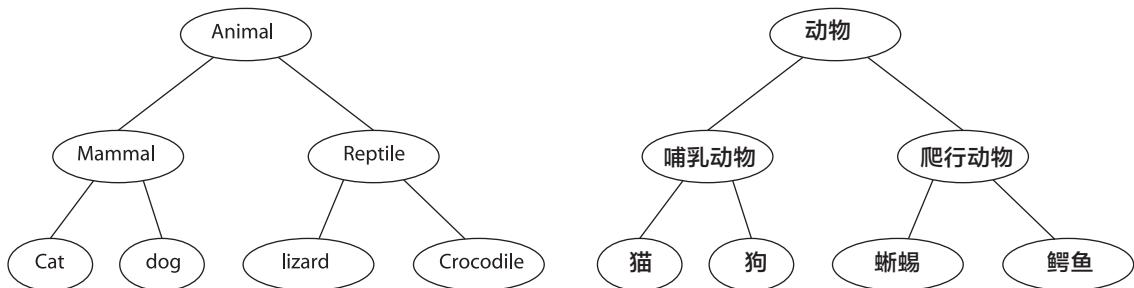


图 22.1

下面是继承的语法

```

class Animal {
    int age;
    Animal() {
        age = 0;
    }
    void eat() {
        // eating code goes here
    }
    void sleep() {
        // sleeping code goes here
    }
}

class Dog extends Animal {
    Dog() {
        super();
    }
    void bark() {
        println("WOOF!");
    }
}
  
```

Animal 类是父类(或者叫超类)

变量 age 还有函数 eat()、sleep() 都被
猫和狗继承。

Dog 是子类。这里用代码“extends Animal”
来表明

```
class Cat extends Animal {
    Cat() {
        super();
    }

    void bark() {
        println("MEOW!");
    }
}
```

super() 意味着执行父类中所能找到的代码

bark()不是父类中的内容，这个我们必须要在子类中明确。

关于新的内容，下面有介绍：

- **extends**—这是用于指示子类引用父类的关键词。注意每个类只能扩展一个类。但是类可以不断的扩展，这就意味着，狗可以扩展动物，小狗可以扩展狗。一行代码就继承了所有的东西。
- **super()**—Super会调用父类中的函数构造器。换句话说，无论你再父类函数构造器中做什么，子类中也会出现相同的内容。

子类可以在父类的基础上添加额外的函数和属性。例如，假设 Dog 对象有一个毛发颜色变量，它可以在函数构造器中随机设置。类就看起来像这样。

```
class Dog extends Animal {
    color haircolor;
    Dog() {
        super();
        haircolor = color(random(255));
    }

    void bark() {
        println("WOOF!");
    }
}
```

子类中能够有不同于父类的其他的额外的变量

注意如何通过 super() 来调用父类函数构造器，age 设定为 0，但是 haircolor 的设定是在 Dog 函数构造器本身。假设一个狗对象 eat 于一般对象不同。父类中的函数也可以在子类中重写覆盖掉。

```
class Dog extends Animal {
    color haircolor;
    Dog() {
        super();
        haircolor = color(random(255));
    }

    void eat() {
        // Code for how a dog specifically eats
    }

    void bark() {
        println("WOOF!");
    }
}
```

但是如果狗应该和动物的 eat 是一样的，但是又有一些额外的不同呢？子类能够同时运行从父类继承中继承的代码以及一些自定义代码。

```
class Dog extends Animal {
    color haircolor;

    Dog() {
        super();
        haircolor = color(random(255));
    }

    void eat() {
        // Call eat() from Animal
        super.eat();
        // Add some additional code
        // for how a dog specifically eats
        println("Yum!!!");
    }

    void bark() {
        println("WOOF!");
    }
}
```

子类能够执行来自父类的函数同时加入自己的一些代码。

练习 22-1：继续我们 Car 的例子，利用封装，看看怎么设计一个汽车类系统(即轿车，卡车，巴士，摩托车)？什么变量和函数应该包含在父类中？子类中应该添加或者覆盖什么内容？如果我们像包含飞机、火车以及轮船应该怎么办？将图 22.1 作为参考。



22.3 一个继承的例子：形状

现在我们已经对继承的理论和语法使用有一个基本的了解，我们可以在 Processing 中开发一个例子。

一个典型就是涉及形状的例子。虽然有一点老生常谈，但是它非常有用，因为特别简单。我们需要创建一个通用的“Shape”类，所有的 Shape 对象都有x,y坐标以及尺寸，并且有函数 display。Shape 通过“jiggling”在屏幕中随机地移动。。

```

class Shape {
    float x;
    float y;
    float r;

    Shape(float x_, float y_, float r_) {
        x = x_;
        y = y_;
        r = r_;
    }

    void jiggle() {
        x += random(-1,1);
        y += random(-1,1);
    }

    void display() {
        point(x,y);
    }
}

```

一个通用的形状不会真的知道怎么去显示。这里可以在子类中重写。

接下来，我们从 Shape 中创建一个子类(称之为 “Square”)。它会从 Shape 中继承实例变量和函数。我们用“Square”的名称写了一个新的函数构造器，并且利用 super() 执行父类的代码。

```

class Square extends Shape {
    // we could add variables for only Square here if we so desire
    Square(float x_,float y_,float r_) {
        super(x_,y_,r_);
    }

    // Inherits jiggle() from parent
    // Add a display method
    void display() {
        rectMode(CENTER);
        fill(175);
        stroke(0);
        rect(x,y,r,r);
    }
}

```

变量继承了父类的变量

如果父类函数构造器中有参数，super() 也需要将这些参数进行传输

square 覆盖了它父类的 display

请注意，如果我们要用 super() 来调用父类的函数构造器，我们必须包含其父类所需的参数。另外，因为我们想让屏幕上显示方形，我们就重写了 display()。即使我们想让方形移动，我们也不需要写 jiggle() 函数，直接从父类引用就好。

如果我们想让 Shape 的子类包含一些额外的函数呢？下面的例子是 Circle 的类，它除了是 Shape 的扩展，也同样包含实例变量来跟踪颜色。(注意这纯粹是为了演示父类的这个功能)。它同样扩展了 jiggle() 函数用来调整尺寸，并且采用了一个新函数来改变颜色。

```

class Circle extends Shape {
    // inherits all instance variables from parent + adding one
    color c;

    Circle(float x_, float y_, float r_, color c_) {
        super(x_,y_,r_); // call the parent constructor
        c = c_; // also deal with this new instance variable
    }

    // call the parent jiggle, but do some more stuff too
    void jiggle() {
        super.jiggle();
        r += random(-1,1);
        r = constrain(r,0,100);
    }

    void changeColor() {
        c = color(random(255));
    }

    void display() {
        ellipseMode(CENTER);
        fill(c);
        stroke(0);
        ellipse(x,y,r,r);
    }
}

```

Circle 的 jiggles 不仅在变化它的尺寸同样也在变化它的x,y坐标。

changeColor() 函数是 Circle 中独有的

为了演示继承可以正常工作，这里是一个编程，做出了一个 Square 对象以及一个 Circle 对象。Shape、Square、Circle 类不会再包含再里面，但是和上面的代码都是相同的。如例 22-1 所示。

例 22-1：继承

```

// Object oriented programming allows us to define classes in terms of other classes.
// A class can be a subclass (aka "child") of a super class (aka "parent").
// This is a simple example demonstrating this concept, known as "inheritance."

```

```

Square s;
Circle c;

void setup() {
    size(200,200);
    smooth();
    // A square and circle
    s = new Square(75,75,10);
    c = new Circle(125,125,20,color(175));
}

void draw() {
    background(255);
    c.jiggle();
    s.jiggle();
    c.display();
    s.display();
}

```

这个草稿包括一个 Circle 对象和一个 Square 对象。在这里没有 Shape 对象。Shape 仅仅是作为继承树系统中的一个部分。

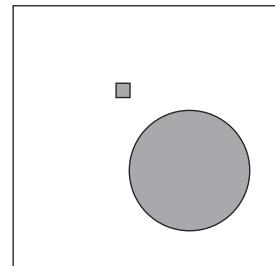


图 22.2

练习 22-2：写一个 Line 的类，继承 Shape 类，并且它拥有2个点的变量。当 line 在移动的时候，这2个点都在动。你可能在 Line 类中不再需要“r”（但是你可不可以加入它来使用？）。



```
class Line _____ {
    float x2,y2;

    Line(_____,_____,_____,_____) {
        super(_____);
        x2 = _____;
        y2 = _____;
    }

    void jiggle() {
        _____
        _____
        _____
    }

    void display() {
        stroke(255);
        line(_____);
    }
}
```



练习 22-3：有没有草稿能够让你使用继承这个概念？找一找并且去修改它。

22.4 多态性

有了继承的概念，我们就可以编程一个多样化的动物世界数组，包括狗，猫，等等。

```
Dog[] dogs = new Dog[100];
Cat[] cats = new Cat[101];
Turtle[] turtles = new Turtle[23];
Kiwi[] kiwis = new Kiwi[6];
```

100条狗, 101只猫, 23只海龟, 6只猕猴。

```

for (int i = 0; i < dogs.length; i++) {
    dogs[i] = new Dog();
}

for (int i = 0; i < cats.length; i++) {
    cats[i] = new Cat();
}

for (int i = 0; i < turtles.length; i++) {
    turtle[i] = new Turtle();
}

for (int i = 0; i < kiwis.length; i++) {
    kiwis[i] = new Kiwi();
}

```

因为每个数组的长度不同，所以我们需要对每个数组做单独的循环

作为一天的开始，动物们都饿了，并且正在寻找吃的。所以吃也要循环。

```

for (int i = 0; i < dogs.length; i++) {
    dogs[i].eat();
}

for (int i = 0; i < cats.length; i++) {
    cats[i].eat();
}

for (int i = 0; i < turtles.length; i++) {
    turtles[i].eat();
}

for (int i = 0; i < kiwis.length; i++) {
    kiwis[i].eat();
}

```

这个工作很庞大，但是我们的世界会包括更多的动物物种。我们要一直写这么多循环是不是有必要呢？毕竟动物都是生物，他们都会吃东西。为什么我们不能有一个 Animal 对象的数组然后为不同种类的动物填充 eat？

```

Animal[] kingdom = new Animal[1000];

for (int i = 0; i < kingdom.length; i++) {
    if (i < 100) kingdom[i] = new Dog();
    else if (i < 400) kingdom[i] = new Cat();
    else if (i < 900) kingdom[i] = new Turtle();
    else kingdom[i] = new Kiwi();
}

for (int i = 0; i < kingdom.length; i++) {
    kingdom[i].eat();
}

```

这是动物类型的数组，但是我们已经放入了 Dog, Cat, Turtle, Kiwi 元素放入其中。

现在是时候让所有的动物都 eat，我们只需要将一个大数组放进循环中即可

Dog 对象的能力是 Dog 类或者 Animal 类的一个部分，这就被称作多态性，这是面向对象编程的第三个概念。

多态性(这个词来自于希腊，意味着许多的形式)是指在多个形式中的单独的对象实例的对待方式。狗肯定是狗，但是因为狗由动物扩展，所以我们可以把狗认为是一个动物。在代码中，我们可以参考 2 种方式。

```

Dog rover = new Dog();
Animal spot = new Dog();

```

通常情况下，左侧类型必须匹配右侧类型。但是由于有多态性，右侧类型可以是左侧类型的子类。

虽然第二行代码看起来似乎违反了语法的规则，但是这两种声明 Dog 对象的方式都是正确的。即使我们声明 spot 作为一个 Animal，我们可以真的做出 Dog 对象，并且将它储存在 spot 变量中。并且我们可以很安全的调用所有的 Animal 函数，因为继承的概念表明 Dog 可以做所有 Animal 能做的事。

如果是 Dog 类，会覆盖 Animal 类中 eat() 函数吗？虽然 spot 声明的时候是作为 Animal，Java 会明确的将其定义为 Dog，并且运行其相应的 eat() 函数。

当我们有数组的时候，这个是非常有用的。

让我们重写之前章节的 Shape 的例子，让它包含许多的 Circle 对象和许多的 Square 对象。

```
// Many Squares and many Circles
Square[] s = new Square[10];
Circle[] c = new Circle[20];
```

老的“没有多态性”的多个
数组方法

```
void setup() {
    size(200,200);
    smooth();
    // Initialize the arrays
    for (int i = 0; i < s.length; i++) {
        s[i] = new Square(100,100,10);
    }
    for (int i = 0; i < c.length; i++) {
        c[i] = new Circle(100,100,10,color(random(255),100));
    }
}

void draw() {
    background(100);
    // Jiggle and display all squares
    for (int i = 0; i < s.length; i++) {
        s[i].jiggle();
        s[i].display();
    }

    // Jiggle and display all circles
    for (int i = 0; i < c.length; i++) {
        c[i].jiggle();
        c[i].display();
    }
}
```

多态性允许我们通过一组包含 Circle 和 Square 对象的 Shape 对象数组来简化上面的代码。我们不用担心哪个是哪个，这些都会有交代。(另外，请注意关于类的代码都没有修改，所以我们这里没有将类的代码加入)如例 22.2 所示。

例 22-2：多态性

```
// One array of Shapes
Shape[] shapes = new Shape[30];

void setup() {
    size(200,200);
    smooth();
    for (int i = 0; i < shapes.length; i++) {
        int r = int(random(2));
        // randomly put either circles or squares in our array
        if (r == 0) {
            shapes[i] = new Circle(100,100,10,color(random(255),100));
        } else {
            shapes[i] = new Square(100,100,10);
        }
    }
}

void draw() {
    background(100);
    // Jiggle and display all shapes
    for (int i = 0; i < shapes.length; i++) {
        shapes[i].jiggle();
        shapes[i].display();
    }
}
```

新的多态性概念方法能够让一个数组包含Shape对象的不同扩展类型。

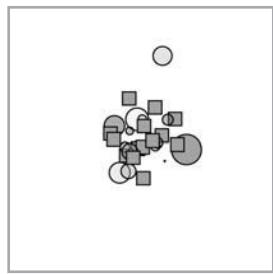


图 22.3



练习 22-4：添加你在练习22-2到22-3所创建的Line类。随机把Circle对象、Square对象和Line对象放入数组中。请注意你的代码应该记不会和上面的例子有太多不同(你应该只需要编辑setup()中的内容)。



练习 22-5：让你练习 22-3中的草稿实现多态性。

22.5 过载

在第16章中，我们学习了如何创建一个Capture对象从摄像头中读取实时图像。如果我们看看Processing reference 页面 (<http://www.processing.org/reference/libraries/video/Capture.html>)，你会注意到Capture函数构造器可以调用3个、4个或者5个参数。

```
Capture(parent, width, height)
Capture(parent, width, height, fps)
Capture(parent, width, height, name)
Capture(parent, width, height, name, fps)
```

函数能够采用不同数量的参数，实际上在第1章我们就已经接触过。例如**fill()**，能够调用一个参数(只能表示灰阶)，或者三个参数(表示RGB色彩)，或者四个参数(RGB颜色及透明度)。

```
fill(255);
fill(255,0,255);
fill(0,0,255,150);
```

能够为相同函数名的函数(但是参数不同)定义功能，这就是过载。例如**fill()**，Processing没有混淆**fill()**的定义，它能够很简单地找到参数进行匹配。函数名称结合参数被称为函数签名—这样能够定义函数的不同。让我们用一个例子说明过载是如何这么有用。

比方说你有一个Fish类，每一个Fish对象都有一个坐标：x和y。

```
class Fish {
    float x;
    float y;
```

如果当你去创造 Fish 对象的时候，你有时可能会想用一个随机的坐标，有时又会想到去指定一个坐标。要达到这一点，我们可以写两个不同的函数构造器。

```
Fish() {
    x = random(0,width);
    y = random(0,height);
}

Fish(float tempX, float tempY) {
    x = tempX;
    y = tempY;
}
```

过载允许我们为相同的对象定义 2 个
函数构造器(只是采用的参数不同)

当你在主程序中创建Fish 对象是，你就可以同时创造他们两个：

```
Fish fish1 = new Fish();
Fish fish2 = new Fish(100,200);
```

如果 2 个构造器都被定义，对象就
能随便使用它们进行初始化了。

23. Java

“I love coffee, I love tea, I love the Java Jive and it loves me.”

—Ben Oakland and Milton Drake

本章内容：

- Processing是真正的Java
- 如果没有Processing我们的代码会是什么样子？
- 探索Java API
- 一些有用的例子：ArrayList 和 Rectangle
- 异常(错误)的处理—try 和 catch
- 除了Processing之外的

23.1 揭示向导

也许，你在一直阅读着这本书的同时，你可能已经在想：“天哪，这好帅。我喜欢编程。我很高兴学习了 Processing。我等不及要去学习Java了！”有趣的是，你不需要真的去学习 Java。把 Processing 的窗帘拉开，我们会发现其实我们一直在学 Java。例如，在 Java 中你可以：

- 用同样的方法声明，初始化，并且使用变量
- 用同样的方法声明，初始化并且使用数组
- 用同样的方法运用条件语句和循环语句
- 用同样的方法定义和调用函数
- 用同样的方法创建类
- 用同样的方法实例化对象

当然 Processing 给予我们提供了一些额外的东西(并且在这里或那里简化了一些东西)，这就是为什么这个工具是用于开发交互式图形项目的强大工具。这里是一个关于 Processing 所提供的内容的列表，这不比 Java 更适合：

- 一套用于绘制形状的函数
- 一套用于读取和显示文字、图片和视频的函数
- 一套用于 3D 转换的函数
- 一套用于鼠标和键盘交互的函数
- 一个简单的开发环境
- 一个拥有艺术家、设计师和程序员的友好论坛

23.2 如果没有Processing，我们的代码会是什么样子？

在第2章中，我们讨论过编制和执行的过程—当你按下播放按钮的时候你的代码就会形成一个图形窗口。这个过程的第一步涉及到将你的 Processing 代码转换成 Java。实际上，当你导出小程序时，同样的过程也会发生，你会注意到“Sketchname.pde”所在的文件夹下同样也会出现一个“SketchName.java”文件。这就是你转换的代码，用转换这个词有点不恰当，因为这个变化非常少。让我们来看一个例子：

```
// Randomly Growing Square
float w = 30.0; // Variable to keep track of size of rect

void setup() {
    size(200, 200);
}

void draw() {
    background(100);
    rectMode(CENTER);
    fill(255);
    noStroke();
    rect(mouseX,mouseY,w,w); // Draw a rect at mouse location
    w += random(-1,1); // Randomly adjust size variable
}
```

这里是我们的 Processing 代码

导出草稿，我们可以打开 Java 文件来看看Java的源代码：

```
// Randomly Growing Square with Java Stuff
import processing.core.*;
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;
import java.util.zip.*;

public class JavaExample extends PApplet {
    float w = 30.0f; // Variable to keep track of size of
    public void setup() {
        size(200, 200);
    }

    public void draw() {
        background(100);
        rectMode(CENTER);
        fill(255);
        noStroke();
        rect(mouseX,mouseY,w,w); // draw a rect at mouse location
        w += random(-1,1); // randomly adjust size variable
    }
}
```

被转换成 Java 代码后有一些新的东西在代码顶部，但是一切基本保持不变

很显然没有什么变化，只是有一些代码被加在了 setup() 和 draw() 的前面和后面。

Import语句—在代码顶部，这里有一组导入语句能够让我们访问某些库。在使用 Processing 库的时候我们已经见过。如果我们使用的是 Java 而不是 Processing，那么每次你写代码的时候就必须要指定所有你需要的库。但是在 Processing 中，假设是一个基于 Java 的库(如 `java.applet.*`)，它都是来自于 Processing (`processing.core.*`)，所以说我们在每个草图中都不会看见这些。

public—在 Java 中，变量、函数以及类都会被分为 “public”/公共 或 “private.”/私有。这是给予代码块的特定访问级别。在 Processing 中我们不用过多的担心这个，但是当你的项目移动到大型 Java 编程中的时候，你就需要重点考虑这个。作为独立程序员，你最常见的可能就是批准或拒绝你的访问，这是为了防止出错的一种手段。在第 22 章中的一些关于封装的例子中我们讨论了这个。

class JavaExample—听起来有些熟悉？Java 是一个真正的面向对象的语言。在 Java 中所编写的没有不是属于类的东西。我们用的 Zoot 类，Car 类，PImage 类等等，但是更重要的是草稿也会被作为一个整体的类。Processing 为我们已经填补了这些需要被用到的类，所以我们在学习编程的时候不用担心什么。

extends PApplet—好吧，当我们读完第 22 章，我们应该很清楚这是什么意思。这只是继承的另一个例子。在这里 JavaExample 类是 PApplet 类的一个子类(或者说 PApplet 是 JavaExample 的父类)。PApplet 是 Processing 开发者所开发的类，并且它可以被扩展，我们在 Processing 能够访问的一些服务就是这个类所带来的— `setup()`, `draw()`, `mouseX`, `mouseY`, 等等。这些代码都是藏在每个 Processing 草稿后面的秘密。

Processing 已经拥有了上面所说的内容，所以我们不用担心有什么问题，同时它还能提供访问 Java 编程语言的这个优点。本章的其余部分会展示我们如何访问完整的 Java API。(在第 17,18 章我们学习字符串的时候我们有简单的使用过访问 API)。

23.3 探索 Java API

当我们学习编程的时候 Processing reference 很快的就变成了我们最好的朋友。Java API 开始的时候我们就会感觉很熟悉。当真正熟悉之后我们也会和它变成好朋友，但是现在，先学习一部分。

我们可以探索完整的 Java 文档，通过访问：

<http://java.sun.com/>

在网页中我们可以点击 API 规范：

<http://java.sun.com/reference/api/index.html>

并选择 Java 的版本。在 Mac 上，Processing 会运行选择好的 Java。在 PC 上，Processing “standard” 下载会自带 1.4.2 版本的 Java (Windows 专家版本允许你安装你自己的 Java 版本)。Java 的版本出现更新的信息，你可以在 processing.org 上找到。

在任何情况下，不同 Java 版本之间是有差异的。对于我们来说，这不用过多的担心，我们可以看看 J2SE 1.4.2 的 API，我们想做的几乎都能做。如图 23.1 所示。

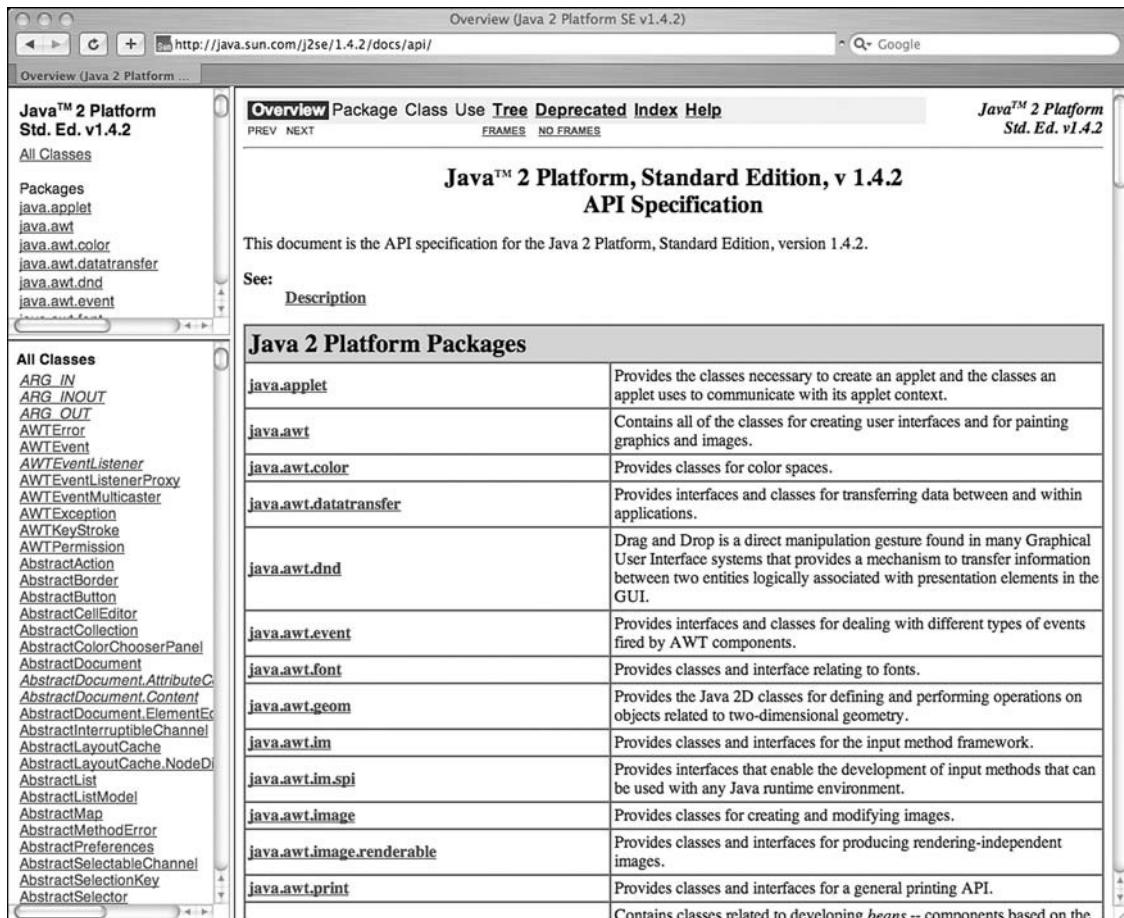


图 23.1 <http://java.sun.com/j2se/1.4.2/docs/api/>

于是很快你就会发现你已经完全被折服了。这很正常，Java API 太巨大了。堆积如山。但这不意味这我们就要完完整整的去细读。它纯粹只是个参考，你可以找到一些你需要的类。

例如，你现在正在做的变成需要高级的随机数，这超越了 random() 的能力范畴，并且你在无意中听说了关于“随机”的类，并且在想“嘿，也许我应该看看这个类！”特定的类可以通过在“All Classes”向下滚动找到你所需要的，或者通过选择合适的包(在这里，合适的包是java.util。一个包，就像一个库，它是类的集合(API通过不同的主题来组织它们)。找类最简单的方法是，在 google 中输入类的名称和 Java(如 “Java Random”)。Java 文档的页面总会是搜索出来的第一个结果。如图 23.2 所示。

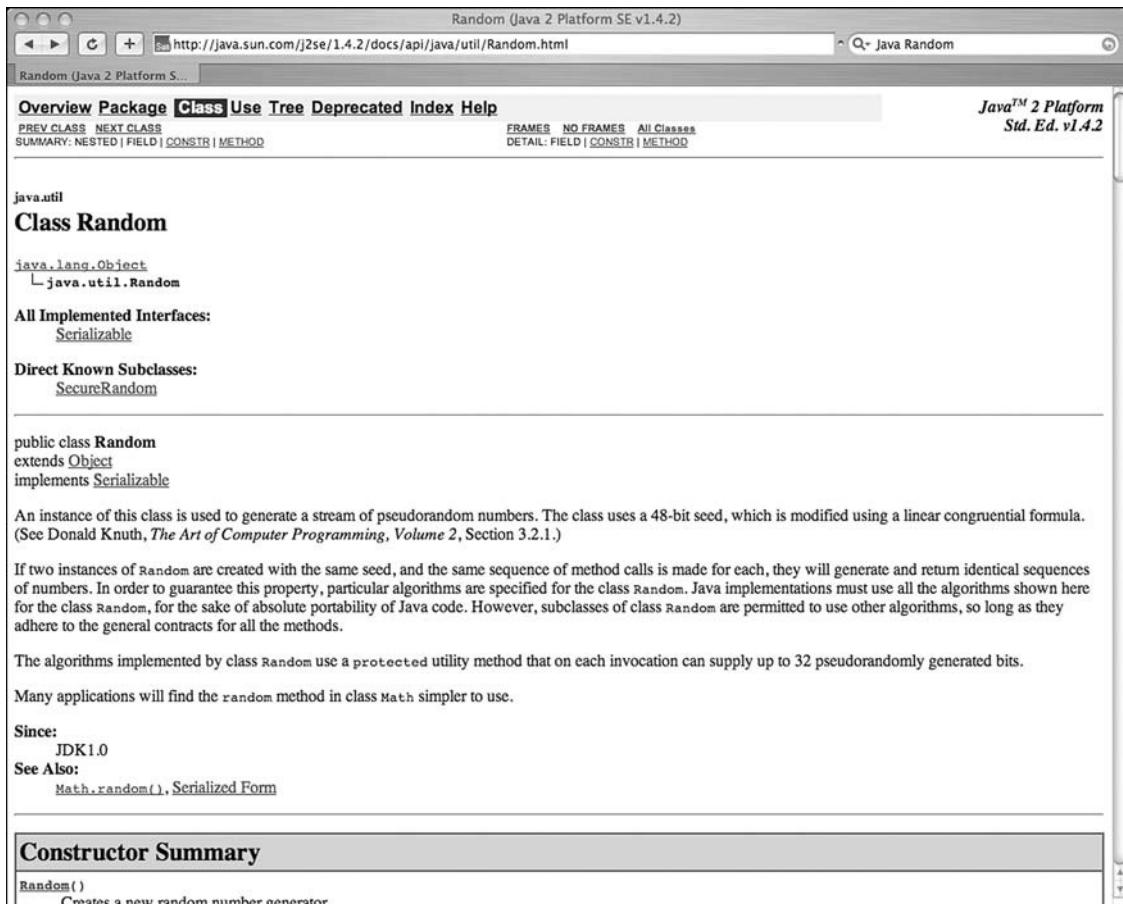


图 23.2

就像 Processing reference 一样，Java 页面也包含了类的解释，函数构造器用来创建一个对象实例，并且拥有变量和函数。因为 `random` 是 `java.util` 包的一部分(在 Processing 中已经导入过)，所以我们不需要写出 `import` 导入语句才能使用它，我们只需要直接用就可以。

下面是一些生成 `random` 对象和调用 `nextBoolean()` 函数的代码，用来判断随机是真或假。

例 23-1：使用 `java.util.Random` 代替 `random()`

```
Random r;
void setup() {
  size(200,200);
  r = new Random();
}
```

声明 `Random` 对象及调用函数构造器可以在<http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html>中找到

```
void draw() {
    boolean trueorfalse = r.nextBoolean();
    if (trueorfalse) {
        background(0);
    } else {
        background(255);
    }
}
```

调用函数请访问：<http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html>



练习 23-1：访问 Java reference 页面找到 Random，并且使用它获取一个0到9之间的随机整数。
<http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html>.

23.4 其他的有用的 Java 类：ArrayList

在第 6 章中，我们学习了如何使用一组数组来保存有序列表信息的轨道。我们创建了一组包含 N 个对象的数据，并且使用“for”循环来访问数组中的每个元素。数组的数量是固定的一我们限制了它有且只有 N 个元素。

当然还有其他方法。一个方法是使用一组非常庞大的数组，并且使用一个变量来跟踪在给定时间所使用数组的数量（详细请回顾第 10 章中雨滴捕手的例子）。Processing 同样也提供 expand()、contract()、subset()、splice() 以及一些用来调整数组的函数。这是非常巨大的，但是有一个类能够实现数组的灵活，允许元素从数组的开始、中间或结尾被删除或者被添加。

这个正是 Java 的类 ArrayList，你可以在 java.util 包中找到。

它的参考页面是：<http://java.sun.com/j2se/1.4.2/docs/api/java/util/ArrayList.html>.

使用 ArrayList 概念上与一个标准的数组类似，但是语法有一些不同。这里的代码演示了数组和 ArrayList 表达的相同结果，第一个是数组，第二个是 ArrayList。这个例子中所有的函数你都可以在 JavaDoc reference 页面找到文档。

```
// 标准的Array方法
int MAX = 10;
//declaring the array
Particle[] parray = new Particle[MAX];

// Initialize the array in setup
void setup() {
    for (int i = 0; i < parray.length; i++) {
        parray[i] = new Particle();
    }
}

// Loop through the array to call methods in draw
void draw() {
    for (int i = 0; i < parray.length; i++) {
```

标准的 Array 方法
这就是我们一直以来的做法，通过索引以及中括号 [] 来访问数组中的元素

```

Particle p = parray[i];
p.run();
p.display();
}

// 新的ArrayList方法
int MAX = 10;
// Declaring and creating the ArrayList instance
ArrayList plist = new ArrayList();

void setup() {
  for (int i = 0; i < MAX; i++) {
    plist.add(new Particle());
  }
}

void draw() {
  for (int i = 0; i < plist.size(); i++) {
    Particle p = (Particle) plist.get(i);
    p.run();
    p.display();
  }
}

```

新的 ArrayList 方法

ArrayList 中的元素通过函数 add() 和 get() 进行添加和访问。所有函数的文档你可以在 Java reference 页面找到：<http://java.sun.com/j2se/1.4.2/docs/api/java/util/ArrayList.html>.

一个对象通过 add() 被添加到 ArrayList 中。

ArrayList 的数量的大小通过 size() 返回一个值。

一个对象通过 get() 可以在 ArrayList 中被访问。你必须“投出”无论 ArrayList 是什么类型，这就是 (Particle). 投出在第 4 章中讨论。

在最后的“for”循环中，我们必须投出对象，把它从 ArrayList 中弄出来。ArrayList 不会持续跟踪对象所存储的类型—尽管这可能是多余的，但是我们的工作就是将类的类型放在括号中(如 Particle)来提醒程序。这个过程就称为投出。

当然，上面的例子有点傻，因为它没有发挥到 ArrayList 的可调整大小的优势，而是使用的一个 10 的固定大小。下面是一个更好的例子，每次循环 draw() 的时候，就会有一个新的 Particle 被添加到 ArrayList。我们还控制了 ArrayList 的大小不会超过 100 个 particle。如图 23.3 所示。

术语“particle system(粒子系统)”在 1983 年由 William T. Reeves 发明。它开发了电影《星际迷航 2：可汗之怒》结尾的“创世纪”效果。

粒子系统是一些独立对象的集合，通常是通过一个简单的形状或点聚集而成。它可以用来模拟自然现象，如爆炸，火灾，烟雾，火花，瀑布，云，雾，花瓣，草，泡沫，等等。

例 23-2：带有 ArrayList 的简单粒子系统

ArrayList particles;

```

void setup() {
  size(200,200);
  particles = new ArrayList();
  smooth();
}

```

```

void draw() {
    particles.add(new Particle());
    background(255);
    // Iterate through our ArrayList and get each Particle
    for (int i = 0; i < particles.size(); i++) {
        Particle p = (Particle) particles.get(i);
        p.run();
        p.gravity();
        p.display();
    }
    // Remove the first particle when the list gets over 100.
    if (particles.size() > 100) {
        particles.remove(0);
    }
}

```

```

// A simple Particle class
class Particle {
    float x;
    float y;
    float xspeed;
    float yspeed;

    Particle() {
        x = mouseX;
        y = mouseY;
        xspeed = random(-1,1);
        yspeed = random(-2,0);
    }

    void run() {
        x = x + xspeed;
        y = y + yspeed;
    }

    void gravity() {
        yspeed += 0.1;
    }

    void display() {
        stroke(0);
        fill(0.75);
        ellipse(x,y,10,10);
    }
}

```

每次通过 `draw()` 循环时，一个新的 `Particle` 对象就会被添加到 `ArrayList` 中。

`ArrayList` 持续跟踪有多少个元素正在储存，并且迭代它们。

如果 `ArrayList` 中有超过 100 个元素，我们会使用 `remove()` 删除第一个元素

练习23-2：重写例23-2，让粒子离开了窗口的时候(即它们的y值大于height等条件)，那个particle(粒子)就会开始被删除。



提示：在Particle类中添加一个函数用来返回布尔值

```
_____ finished() {
    if (_____){
        return _____;
    } else {
        return false;
    }
}
```

提示：为了能够正常工作，你必须迭代ArrayList中向后的元素！为什么？因为当元素被删除之后，所有的后续元素都会被移到左侧(如图23.4所示)。

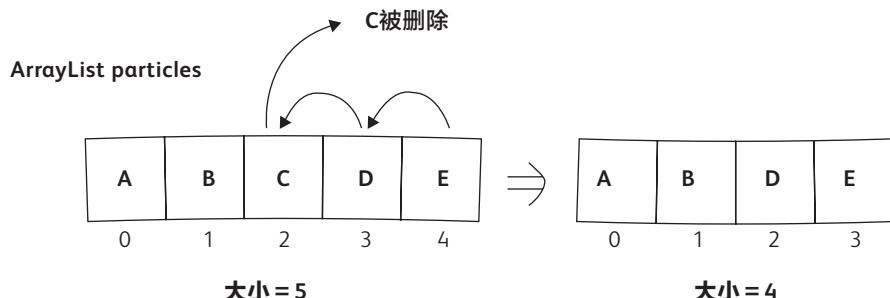


图 23.4

```
for (int i = _____; i _____; i _____) {
    Particle p = (Particle) particles.get(i);
    p.run();
    p.gravity();
    p.render();
    if (_____){
        _____;
    }
}
```

23.4 其他的有用的Java类：Rectangle

第二个有用的Java类是Rectangle类：

<http://java.sun.com/j2se/1.4.2/docs/api/java.awt/Rectangle.html>.

Java Rectangle指定了坐标空间中的一块封闭区域，它通过Rectangle对象的左顶点(x,y)以及宽度和高度来确定。听起来很熟悉？当然，Processing函数rect()可以使用完全相同的参数绘制一个矩形。Rectangle用封装的方法将矩形放进了一个对象。

Java Rectangle有一些有用函数，如contains()。contains()提供了一个简单的方法来检查一个点或者一个矩形是否在那个指定的矩形中，通过接收到的x和y坐标，最后并返回一个基于这个点是否在矩形中的真或假的答案。

这里是一个简单的通过Rectangle对象和contains()实现的翻转的粒子。如例23.3。

例 23-2：带有ArrayList的简单粒子系统

Rectangle rect1, rect2;

```
void setup() {
    size(200,200);
    rect1 = new Rectangle(25,25,50,50);
    rect2 = new Rectangle(125,75,50,75);
}
```

```
void draw() {
    background(255);
    stroke(0);

    if (rect1.contains(mouseX,mouseY)) {
        fill(200);
    } else {
        fill(100);
    }
}
```

```
rect(rect1.x, rect1.y, rect1.width,rect1.height);

// Repeat for the second Rectangle
// (of course, we could use an array or ArrayList here!)
if (rect2.contains(mouseX,mouseY)) {
    fill(200);
} else {
    fill(100);
}
rect(rect2.x, rect2.y, rect2.width,rect2.height);
}
```

这个草稿使用了2个Rectangle对象。关于函数构造器的参数(x,y,width,height)可以在 Java reference 的文档中找到：<http://java.sun.com/j2se/1.4.2/docs/api/java.awt/Rectangle.html>.

contains()函数用来判断鼠标的位置是否位于矩形里面。

Rectangle对象值需要知道这个矩形的相关变量。但是它不能显示。所以我们使用Processing的rect()函数组合Rectangle数据。

让我们来做一些有趣的事，结合ArrayList的“粒子”例子和Rectangle的“翻转”例子。在例23-4中，例子每一帧都会生成并且，通过重力向窗口底部运动。如果它们运动到一个矩形中，那么它们就会被捕获。粒子被存储在ArrayList中，Rectangle对象用来判断它们是否被捕获。(这个例子包含了练习23-2的答案)

例 23-2：带有ArrayList的简单粒子系统

```

// Declaring a global variable of type ArrayList
ArrayList particles;
// A "Rectangle" will suck up particles
Rectangle blackhole;

void setup() {
    size(200,200);
    blackhole = new Rectangle(50,150,100,25);
    particles = new ArrayList();
    smooth();
}

void draw() {
    background(255);
    // Displaying the Rectangle
    stroke(0);
    fill(175);
    rect(blackhole.x, blackhole.y, blackhole.width,blackhole.height);
    // Add a new particle at mouse location
    particles.add(new Particle(mouseX,mouseY));
    // Loop through all Particles
    for (int i = particles.size() - 1; i >= 0; i--) {
        Particle p = (Particle) particles.get(i);
        p.run();
        p.gravity();
        p.display();
        if (blackhole.contains(p.x,p.y)) {
            p.stop();
        }
        if (p.finished()) {
            particles.remove(i);
        }
    }
}

// A simple Particle Class
class Particle {
    float x;
    float y;
    float xspeed;
    float yspeed;
    float life;

    // Make the Particle
    Particle(float tempX, float tempY) {
        x = tempX;
        y = tempY;
        xspeed = random(-1,1);
        yspeed = random(-2,0);
        life = 255;
    }

    // Move
    void run() {
        x = x + xspeed;
        y = y + yspeed;
    }
}

```

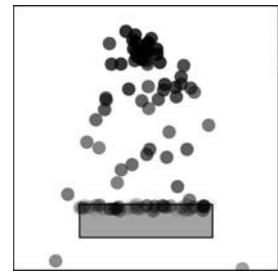


图 23.6

如果Rectangle包含了粒子的坐标，那么
粒子停止运动

```

// Fall down
void gravity() {
    yspeed += 0.1;
}

// Stop moving
void stop() {
    xspeed = 0;
    yspeed = 0;
}

// Ready for deletion
boolean finished() {
    life -= 2.0;
    if (life < 0) return true;
    else return false;
}

// Show
void display() {
    stroke(0);
    fill(0,life);
    ellipse(x,y,10,10);
}
}

```

粒子对象有一个“life”的递减变量，当这个值减小到0，那么粒子可以从ArrayList中移出



练习23-3：编写一个Button类，让它包含Rectangle对象用来判断鼠标点击是否在按钮里面。(这时练习9-8的延伸)

23.4 异常(错误)处理

除了有一个庞大的变成库以外，Java编程语言还包含了一些学习Processing时没有接触过的能力。现在我们就要探索这些特殊能力之一的：异常处理。

编程发生错误，我们都见过它们。

```

java.lang.ArrayIndexOutOfBoundsException
java.io.IOException: openStream() could not open file.jpg
java.lang.NullPointerException

```

当这些错误发生的时候，真是让人心烦。真的就是这样。错误信息显示出来，程序就没有继续运行了。也许你已经开发了一些技术来防止这些错误发生。例如：

```
if (index < somearray.length) {
    somearray[index] = random(0,100);
}
```

以上是一个“检查错误”的方式。代码使用了一个条件语句在元素访问索引之前确保索引是否有效。上面的代码很严谨，我们也应该如此小心。

不是所有的情况都是这么容易去避免的，所以这就有了异常处理功能的用武之地。异常处理指的是在一般事件中发生错误的编程时处理异常的过程。

代码结构的异常处理在Java中被称为尝试捕捉。换句话说，“尝试运行一些代码，如果出现了问题，捕捉这个出错的地方然后再运行其他代码。”如果错误已经捕捉到，程序就可以继续运行。让我们看看我们如何使用尝试捕捉风格检查代码并改写数组索引错误。

```
try {
    somearray[index] = 200;
} catch (Exception e) {
    println("Hey, that's not a valid index!");
}
```

在“try”中的代码可能会出现错误。如果代码发生错误，它就会进入“catch”。

上面的代码能够捕捉到任何可能会发生的异常。捕捉的类型是普通异常。如果你想执行基于指定异常以外的代码，你可以学习以下代码中的方式：

```
try {
    somearray[index] = 200;
} catch (ArrayIndexOutOfBoundsException e) {
    println("Hey, that's not a valid index!");
} catch (NullPointerException e) {
    println("I think you forgot to create the array!");
} catch (Exception e) {
    println("Hmmm, I dunno, something weird happened!");
    e.printStackTrace();
}
```

不同的“catch”会捕捉不同类型
的异常。此外，每个异常都是
一个对象，因此可以有函数来
调用它们。例如：

`e.printStackTrace();`

显示异常的详细信息。

然而上面的代码不能显示什么自定义的错误消息。有的时候我们需要的不仅仅是一个解释。例如，第18章中的例子我们会从URL路径加载数据。如果我们的草稿与URL连接失败呢？它会崩溃并退出。对于异常处理，我们可以捕捉错误并且手动填写一袋字符串的数组让草稿能够继续运行。

```

String[] lines;
String url = "http://www.rockinurl.com";
try {
    lines = loadStrings(url);
} catch (Exception e) {
    lines = new String[3];
    lines[0] = "I could not connect to " + url + " !!! ";
    lines[1] = "But here is some stuff to work with ";
    lines[2] = "anyway. ";
}
println(lines);

```

如果发生连接URL的问题，我们只需要填补了带有虚拟文本的数组，以便草图仍然可以运行。

23.4 Processing之外的Java

在这里，这是书的最后一节。如果你想你可以休息一下。出去外面走一走甚至慢跑。

OK,现在回来乐？

太好了，让我们继续。

当我们学完了这本书，我们将来看看最后一个话题：做什么以及什么时候应该去开始Processing之外的代码。

但是我们非要这样做吗？

在没有涉及人和图形应用的程序的情况下，利用其他编程环境是值得的。也许你需要写一个关于财务信息的程序，并让它纪录在数据库中。或者编写一个后台聊天服务器。虽然这些编程能够在Processing环境下完成，但是它们不是必须需要Processing来运行的。

在开发越来越大的项目并且涉及许多类的时候，Processing环境会变的不是很好用。例如你的项目涉及20个类。你需要创建20个标签来管理这些类，那更多是不是会更麻烦？在这种情况下，专为大型Java项目所创立的开发环境会更好。由于Processing就是Java，所以你仍然可以在其他的开发环境中倒入它的核心库来进行使用。

所以你会怎么做？首先，我想说你不要着急。享受Processing并且让自己慢慢熟悉这代码风格，在Java中很多的问题会随之而来。

如果你觉得你准备好了，第一步只需要你花一些时间来浏览Java的网站：

<http://java.sun.com/>

开始教程之一

<http://java.sun.com/docs/books/tutorial/>

这个教程会覆盖这本书中相同的资料，但是这从一个纯Java的角度来说明的。

下一步是尝试编写并运行一个“Hello World”的Java程序。

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World. I miss you Processing.");
    }
}
```

Java网站有教程来解释所有上面所写的编程含义，并且提供在“命令行”编写和运行它的指令。

<http://java.sun.com/docs/books/tutorial/getStarted/index.html>

最后，如果你想深入编写，赶快去下载Eclipse:

<http://www.eclipse.org/>

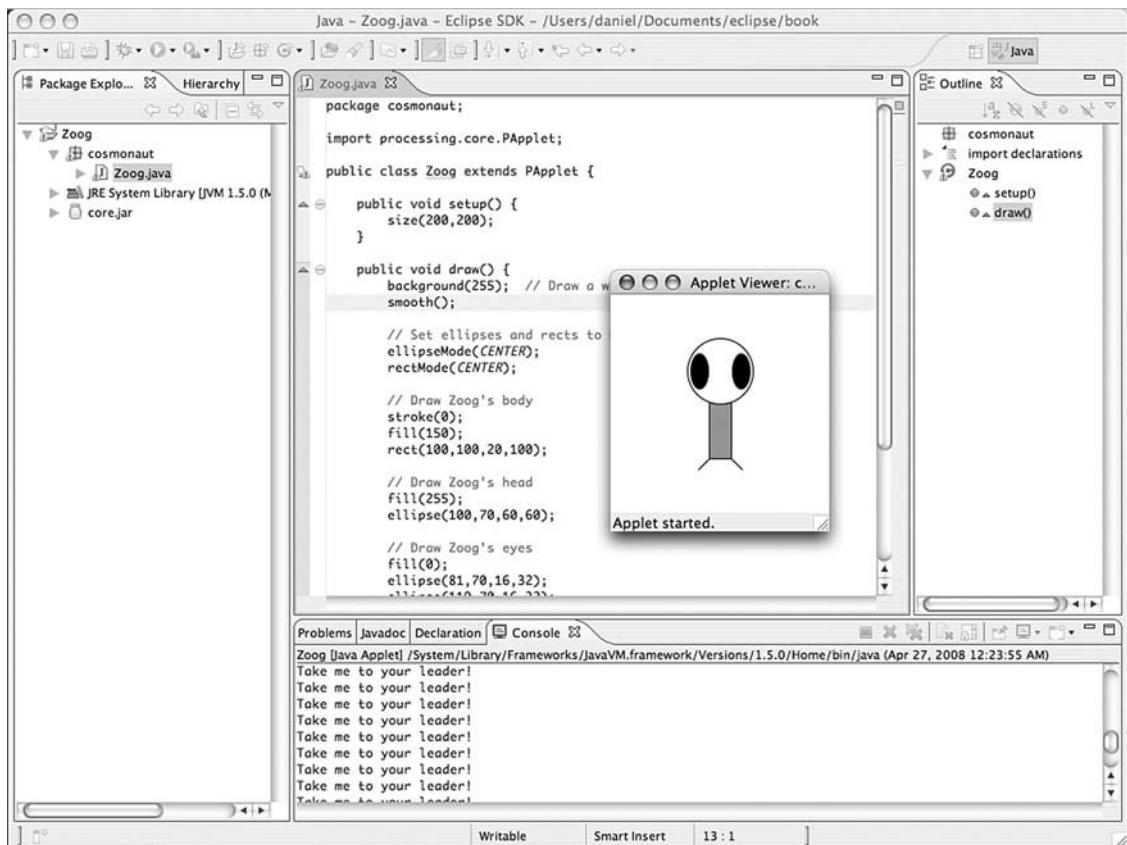


图 23.7

Eclipse是一个Java的开发环境，它拥有很多先进功能。有些功能也会让你头疼，并且有些功能会让你想起Eclipse时候的编程。请记住，当你在第2章开始学习Processing的时候，你运行一个草图的时间可能就不到5分钟。但是如果是在Eclipse中，你可能需要很长的时间去上手。如图23.7所示。

关于如何在Eclipse环境中运行Processing草稿，请访问这本书的网站查看教程。

感谢阅读。反馈是我最好的鼓励和赞赏。

附录：常见错误

错误	页数
unexpected token: _____	440
Found one too many { characters without a } to match it.	440
No accessible field named “myVar” was found in type “Temporary_#####_#####”	441
he variable “myVar” may be accessed here before having been definitely assigned a value.	442
java.lang.NullPointerException	442
Type “Thing” was not found.	444
Perhaps you wanted the overloaded version “type function (type \$1, type \$2, ...);” instead?	445
No method named “fnction” was found in type “Temporary_#####_#####”. However, there is an accessible method “function” whose name closely matches the name “fnction”.	445
No accessible method with signature “function (type, type, ...)” was found in type “Temporary_3018_2848”.	445
java.lang.ArrayIndexOutOfBoundsException	446

错误

```
unexpected token: something
```

翻译：

我在这里需要一段代码，这最有可能是一些标点符号，就像一个分好，括号等等。

这种错误通常是打错字造成的。不幸的是，代码行之间很容易产生错误点。“something”有时候是对的，真正的错误有时候是由于代码行之间丢失或保函错误的标点符号。下面是一些例子：

错误	正确
int val = 5	int val = 5;
if (x < 5 { ellipse(0,0,10,10); }	if (x < 5) { ellipse (0,0,10,10); }
for (int i = 0, i < 5,i ++) { println(i); }	for (int i = 0; i < 5; i ++) { println(i); }

错误

Found one too many { characters without a } to match it.

翻译：

你忘记闭合代码快，如在if语句，loop循环的时候忘记的括号。

在任何时候，你如果有一个正大括号(“{”)，你就必须有一个与之相匹配的反大括号(“}”)。由于代码块之间经常互相嵌套，所以就会很容易忘记反大括号而出现这个错误。

错误	正确
void setup() { for (int i = 0; i < 10; i ++) { if (i > 5) { line(0,i,i,0); println(„ „); } }	void setup() { for (int i = 0; i < 10; i ++) { if (i > 5) { line(0,i,i,0); println(„ „); } } }

错误

No accessible field named "myVar" was found in type "Temporary_7665_1082"

翻译：

我不知道有变量” myVar ”，它没有在我能看见的地方进行声明。

如果你没有声明变量就会发生这种错误。记住，你不能在你声明变量的时候缺少它的类型。如果你像下面这样写，你就会得到错误：

```
myVar = 10;
```

正确写法：

```
int myVar = 10;
```

当然，你一次只能声明一次变量，否则你就会出现问题。但是下列这种情况好像没问题。

```
int myVar = 10;  
myVar = 20;
```

OK, 变量被声明

如果你使用一个局部变量而在代码块外面使用它同样也会出现问题。例如：

```
if (mousePressed) {  
    int myVar = 10;  
}  
ellipse(myVar,10,10,10);
```

错误！ myVar 在这里是 if 语句中的局部变量，所以不能在外面访问使用。

这里是一个修正版：

```
int myVar = 0;  
if (mousePressed) {  
    myVar = 10;  
}  
ellipse(myVar,10,10,10);
```

在 if 语句外面声明变量是可以的

如果你在 setup() 中声明变量，但是尝试在 draw() 中去使用。

```
void setup() {  
    int myVar = 10;  
}  
void draw() {  
    ellipse(myVar,10,10,10);
```

错误！ myVar 在这里是 setup 中的局部变量，所以不能在外面访问使用。

修正版：

```
int myVar = 0;  
void setup() {  
    myVar = 10;  
}  
void draw() {  
    ellipse(myVar,10,10,10);
```

如果你使用数组，同样不允许出现上面的错误

```
myArray[0] = 10;
```

错误！没有数组被声明。

错误

The variable “ myVar ” may be accessed here before having been definitely assigned a value.

翻译：

你声明了变量“myVar”，但是没有初始化它。你应该先给它一个值。

这是一个非常容易修复的错误。最有可能的就是你忘记给你声明的变量一个初始默认值。这种错误只会发生在局部变量(全局变量不需要关心这些，只需要假定他的值为0，要么它就会抛出错误NullPointerException)。

```
int myVar;  
line(0, myVar, 0, 0);
```

错误！myVar没有一个初始化值。

```
int myVar = 10;  
line(0, myVar, 0, 0);
```

OK！myVar等于10。

如果你在变量还没有被分配值的时候尝试使用一个数组同样也会出现错误

```
int[] myArray;  
myArray[0] = 10;
```

错误！myArray没有正确的创建个初始化值。

```
int[] myArray = new int[3];  
myArray[0] = 10;
```

错误！myArray是一个三个整数的数组。

错误

java.lang.NullPointerException

翻译：

你声明了变量“myVar”，但是没有初始化它。你应该先给它一个值。

NullPointerException错误可能是最难修复的问题之一。它最常见的错误方式是忘记初始化对象。正如第8章中，当你声明一个对象变量的时候，首先应该给予一个值是空的，就代表啥都没有。(这并不适用于原始类，如整数和浮点数)如果你尝试不初始化去使用这个对象变量(调用他的函数构造器)，就可能会发生错误。这里有一些例子(假设一个存在的类叫“Thing”)。

```
Thing thing;
void setup() {
}
```

错误！“thing”从来没有被初始化过，所以他的值为无效。

```
void draw() {
    thing.display();
}
```

更正版：

```
Thing thing;
void setup() {
    thing = new Thing();
}

void draw() {
    thing.display();
}
```

OK！“thing”不是无效，因为他在函数构造器中被初始化了。

有时候如果你已经初始化了对象，但是因为是局部变量的原因也会发生错误：

```
Thing thing;
void setup() {
    Thing thing = new Thing();
}

void draw() {
    thing.display();
}
```

错误！这里的代码的声明以及初始化是一个完全不同的“thing”变量(虽然它们的名字相同)。这个局部变量只针对setup()。全局变量“thing”仍然为无效。

同样的数组也是如此。如果你忘记初始化元素，你就会错误：

```
Thing[] things = new Thing[10];
for (int i = 0; i < things.length; i++) {
    things[i].display();
}
```

错误！数组中的所有元素都无效。

更正版：

```
Thing[] things = new Thing[10];
for (int i = 0; i < things.length; i++) {
    things[i] = new Thing();
}

for (int i = 0; i < things.length; i++) {
    things[i].display();
}
```

OK！第一个循环初始化了数组中的所有元素

最后，如果你忘记给你的数组分配数量，同样会出现这个问题

```
int[] myArray;
void setup() {
    myArray[0] = 5;
}
```

错误！myArray是无效的，因为没有创建它并给它数量。

更正版：

```
int[] myArray = new int[3];
void setup() {
    myArray[0] = 5;
}
```

OK! myArray的数量为三个。

错误

Type “Thing” was not found.

翻译：

你试图声明一个Thing类型的变量，但是我不知道Thing是什么。

这是错误的，因为你可能是忘记定义一个叫“Thing”的类了，或者是写错了一个变量类型。

错别字很常见的就是：

```
intt myVar = 10;
```

错误！你需要输入的类型可能是“int”而不是“intt”。

有时候，如果你初始化对象但是作为局部变量，也会出现相同的错误

```
Thing myThing = new Thing();
```

错误！如果你没有定义一个叫“Thing”的类。

如果你这样写就是对的：

```
void setup() {
    Thing myThing = new Thing();
}
class Thing {
    Thing() {
    }
}
```

OK! 你定义了一个叫“Thing”的类。

最后如果你忘记导入库而去使用这个库的对象，那么同样会发生错误：

```
void setup() {
    Capture video = new Capture(this,320,240,30);
}
```

错误！你忘记导入了
video库。

更正版：

```
import processing.video.*;
void setup() {
    Capture video = new Capture(this,320,240,30);
}
```

OK! 你导入了video库。

错误

Perhaps you wanted the overloaded version “type function (type \$1, type \$2, ...);” instead?

翻译：

您所调用的函数不正确，但是我想我知道你想要说什么。你想要调用函数使用这些参数吗？

发生这种错误大多数可能就是你填写错了参数的个数。比如，你想绘制一个圆，你需要一个x坐标、y坐标、宽度和高度，但是你却这么写：

ellipse(100,100,50);

错误！ellipse()需要四个参数。

你会得到错误：“Perhaps you wanted the overloaded version ‘void ellipse(float \$1, float \$2, float \$3, float \$4);’ instead?”这个错误列出了你对函数的错误定义，说明你需要四个参数，所有的数都是浮点数。如果你写错了参数的类型，同样也会发生错误。

ellipse(100,100,50, “Wrong Type of Argument ”);

错误！ellipse()参数不能使用字符串类型。

错误

No method named “fnction” was found in type “Temporary_9938_7597”. However, there is an accessible method “function” whose name closely matches the name “fnction”.

翻译：

您在调用一个我从来没有听过的函数，我想我知道你想说什么，因为我看到一个函数和您所写的类似。

当你写错了函数的名称时就会发生这种错误：

elipse(100,100,50,50);

错误！你输入了错别字“ellipse”。

错误

No accessible method with signature “function (type, type, ...)” was found in type “Temporary_3018_2848” .

翻译：

您在调用一个我从来没有听过的函数，我不知道您想要表达什么！

如果您想调用的函数时完全不存在的，并且Processing也猜不到这是一个什么样的函数，就会出现错误：

functionCompletelyMadeUp(200);

错误！除非你已经定义了这个函数，要不然 Processing没有办法知道这个函数是什么。

同样，函数需要在对象上被调用

```
functionCompletelyMadeUp(200);
```

错误

java.lang.ArrayIndexOutOfBoundsException: ##

翻译：

您试图访问数组中不存在的元素。

当任何数组中的索引值无效时就会发生这种错误。例如，如果你的数组数量是10，有效的索引值是0到9。任何小于0或者大于9的索引值都产生错误。

```
int[] myArray = new int[10];
myArray[-1] = 0
myArray[0] = 0;
myArray[5] = 0;
myArray[10] = 0;
myArray[20] = 0;
```

错误！-1不是有效的索引值

OK！0~5是有效的索引值

错误！10不是有效的索引值

错误！20不是有效的索引值

当你在索引中使用变量时这个错误可能有些不太好调试。

```
int[] myArray = new int[100];
myArray[mouseX] = 0
```

错误！mouseX会超过99

```
int index = constrain(mouseX,0,myArray.length-1);
```

```
myArray[index] = 0;
```

OK！mouseX被限定在0~99之间

一个循环也同样是问题的根源。

```
for (int i = 0; i < 200; i++) {
    myArray[i] = 0;
}
```

错误！循环会超过99

```
for (int i = 0; i < myArray.length; i++) {
    myArray[i] = 0;
}
```

OK！使用数组的数量作为条件循环

```
for (int i = 0; i < 200; i++) {
    if (i < myArray.length) {
        myArray[i] = 0;
    }
}
```

OK！如果你的循环真的需要到200，那么你可以在循环中嵌套一个if语句。

索引

- AIFF (Audio Interchange File Format) 380, 382
- Algorithms 165
- Act II, getting ready for 188–9
- catcher, 168–70
- from ideas to parts 167–8
- intersection 170–5
- project 189
- puttin' on the Ritz 182–8
- raindrops 178–82
- timer 176–8
- Alpha transparency 14
- Angles 210–11
- Animation:
 - with image 255–6
 - of text 309–12
- API (Application Programming Interface) 349
- Append() function 157, 158
- Apple Computers 273
- Applets 20
- Arc() 8, 9
- Arduino web sites 369
- Arguments 25, 108–13
- ArrayList 426–9
- ArrayOfInts 145
- Arrays 141
- concepts 144–5
- declaration and creation of 145–6
- of images 258–9
- initialization 146–7
- interactive objects 154–7
- of objects 152–4
- operations 144, 147–9
- processing's array functions 157–9
- snake 150–2
- Zoogs in 159–60
- ASCII (American Standard Code for Information Interchange) code 364, 370
- Asterisk 363
- Asynchronous requests 339–42
- vs. synchronous requests 353–4
- Audacity 382
- Available() function 275, 355
- Background removal 292–4
- Background() function 11, 36, 195
- BeginRecord() function 398
- BeginShape() function 231, 233
- Bit 10
- BlobDetection 298
- Blobs 298
- Block of code 32
- Boolean expressions 61
- Boolean variables 71–4
- Bouncing ball 74–8
- Box() function 235
- Brightness mirror 284–6
- Brightness() function 288
- Broadcasting 360–2
- Bubble class constructor 329
- Bug 191
- Built-in libraries 196
- Byte 10
- Callback function 276, 340
- Capture constructor 275
- Capture object 274–5
- CaptureEvent() function 276, 340, 355
- Cartesian coordinates 212
- Casting 57, 427
- Catcher 168–70
- Caught() function 185
- "CENTER" mode 6, 7
- Change() function 331
- Characters 47
- CharAt() function 304, 316, 323
- Charcount 313
- Clapper 390
- Class 122, 407
- Class JavaExample 423
- Class name 125
- Client:
 - creation 358–9
 - multi-user communication 366–7
- Codec 402
- Coding, in processing 20–3
- Color selector 13
- Color transparency 14–15
- ColorMode() function 15, 257
- Compilation 26
- Computer vision 287–91
- libraries 297–8
- Concat() function 157

Concatenation 258 , 306 , 324
 Conditionals:
 boolean expressions 61
 boolean variables 71–4
 bouncing ball 74–8
 description 65–7
 if, else, else if 62–4
 logical operators 67–9
 multiple rollovers 69–70
 physics 101 , 79–81
 Constrain() function 65 , 66 , 89
 Constructor 125
 arguments 131–4
 Contains() function 429 , 430
 Contributed libraries 196–8
 Control structure 84
 Convolution() function 269
 Copy() 292 , 304
 “CORNER” mode 6 , 7
 Cos() function 213
 Cosine 212 , 213
 CreateFont() function 308
 CreateImage() function 255
 Curve vertex 233
 Curve() 8 , 9
 Custom color ranges 15–16
 Dampening effect 80
 Data 125
 types 47
 Data input 323
 asynchronous requests 339–42
 sandbox 352
 splitting and joining 324–7
 string manipulation 323–4
 text analysis 338–9
 text files, reading and writing 327–33
 text parsing 333–8
 XML 342–6
 XML library, processing 346–9
 Yahoo API 349–52
 Data streams 353
 broadcasting 360–2
 client creation 358–9
 multi-user communication 362
 client 366–7
 server 362–5
 server and client 368
 serial communication 368–71
 with handshaking 371–2
 with strings 372–4
 server creation 354–7
 synchronous vs. asynchronous request 353–4
 Debugging 191
 human being, involvement of 191
 println() 193–4
 simplify 192–3
 taking a break 191
 Decimal numbers 47
 Delimiter 325
 Descartes, René 212
 Display() function 107 , 108 , 155 , 193 , 230 , 245
 Displaying text 306–9 , 315–21
 Dist() function 116 , 117 , 172
 Do-while loop 85
 Doorbell 382
 with Minim 385–6
 with Sonia 382–4
 Dot syntax 195 , 127
 Double-buffering process 36
 Double-threshold algorithm 391
 Draw() function 32–4 , 36 , 37 , 38 , 72 , 75 , 90 , 93 , 94 , 96 , 103 , 105 , 107 , 110 , 114 , 123 , 179 , 183 , 227 , 255 , 307 , 310
 Draw() loop 51 , 150 , 275
 DrawBlackCircle() function 105 , 109
 DrawCircle() 218
 DrawSpaceShip() function 104
 DrawZoog() function 118
 Duration() function 281
 Eclipse 434 , 435
 Ellipse 5
 with variables 55
 Ellipse() function 24 , 109 , 227 , 230
 Encapsulation 407–40
 EndRaw() function 399
 EndRecord() function 398 , 399
 EndShape() function 231 , 233
 Equals() function 305 , 323
 Error 23–4
 exception handling, in Java 432–3
 messages 437–44
 Exit conditions 88–90
 Expand() function 157
 Export Application 395–6
 Exporting 395
 high-resolution PDFs 397–400
 images/saveFrame() 400–1
 MovieMaker 401–3
 stand-alone applications 395–7
 web applets 395
 Extends PApplet 423
 Factorial 217
 Fil() function 104

Fill() function 10, 11, 256, 308
 Filter() function 266
 FinishMovies() function 402
 Flow 31–2
 “For” loop 90–3
 Fractals 217
 FrameRate() function 40
 Functionality, of objects 125
 Functions 103
 arguments 108–13
 definition 105–6
 passing a copy 113–14
 return type 114–17
 simple modularity 106–8
 splitting of 103–4
 user defined 104
 Zoog reorganization 118–19
 Function’s signature 419
 GetChild() function 347
 GetChildCount() function 347, 348
 GetChildren() function 347
 GetContent() 347
 GetElementArray 346
 GetElementAttributeText() 345
 GetElementText() function 345
 GetFloatAttribute() 347
 GetIntAttribute() 347
 GetStringAttribute() 347
 GetSummaries() 350
 Getters 408
 GetTotalResultsAvailable() 351
 Global variable 93–5
 Globs see Blobs
 Good friends 32–4
 Graph paper 3–4
 Graphical user interface (GUI) 71
 Grayscale color 10–12
 Grayscale image 221, 222
 Handshaking, serial communication with 371–2
 High-resolution PDFs 397–400
 Highlight() function 175
 Home() function 318
 HTML 333, 335
 XMLHttpRequest object 340, 341
 Hypertext Transfer Protocol (HTTP) 353
 If, else, else if conditionals 62–4
 Image() function 254, 255, 263
 Imageindex 259
 Images 253
 adjusting brightness 264–5
 animation with 255–6
 array of 258–9
 creative visualization 270–2
 getting started 253–5
 group processing 267–70
 image filtering 256–7
 pixels 260
 displaying of 263
 PImage object 265–6
 setting of 260, 261–2
 processing of 262–4
 sequence 259
 swapping of 259
 tint() 264–5
 Images/saveFrame() 400–1
 Immutable object 305
 Import statements 195, 422–3
 Increment/decrement operators 91
 Index variable 311
 IndexOf() function 323, 334
 Infinite loops 88, 89
 Inheritance 410–13
 Int() function 57
 Integration 83–5
 definition 85
 Interaction 31
 flow 31–2
 good friends 32–4
 mouse, variation with 34–8
 mouse clicks and key presses 38–40
 Interactive strips 155–6
 Interactive Zoog 40
 Intersect() function 170, 174, 184, 185
 Invisible Line of Code 35–6
 IsFinished() function 178
 IsPlaying() function 384
 JAR file 26
 Java 421
 API, exploring 423–6
 code, translation 421–3
 coding outside of processing 433–5
 exception (error) handling 432–3
 Java classes 426
 ArrayList 426–9
 Rectangle 429–32
 wizard, revealing 421
 Java 2D 230
 Java byte code 26
 Java Virtual Machine 26
 Jiggle() function 414
 JiggleZoog() function 118
 JMyron 298

Join() function 325, 326, 327
 Jump() function 281
 Key presses 38–40
 KeyPressed() function 39, 93
 Keywords see Reserved words
 Length, of array 149
 Length() function 304, 323, 334
 Lerp() function 321
 Lib 396
 LibCV 298
 Libraries:
 built-in libraries 196
 contributed libraries 196–8
 defi nition 195
 Line() function 24, 104, 195, 227, 260
 Live video 101, 274
 LiveInput 388–90
 LoadFont() function 307
 LoadImage() function 254, 255
 LoadPixels() function 261, 263, 304
 LoadStrings() function 327, 339
 Local variable 93–5
 “Logical and” 68
 Logical operators 67–9
 “Logical or” 68
 Loop() function 114, 279–80
 Loops 83
 exit conditions 88–90
 “for” loop 90–3
 global variable 93–5
 iteration 83–5
 local variable 93–5
 loop inside the main loop 95–7
 “while” loop 85–8
 Zoog grows arms 97–9
 MakeRequest() 345
 Mandelbrot set 216
 Mathematics 201
 angles 210–11
 event probability, in code 205–7
 modulus 202–3
 oscillation 214–16
 Perlin noise 207–10
 probability review 204–5
 and programming 201–2
 random numbers 203–4
 recursion 216–20
 trigonometry 212–14
 two-dimensional arrays 220–4
 Matrix, defi nition of 241
 Methods see Functions
 Microphone, as sound sensor 388
 Millis() function 176
 Minim library 379, 381
 Mirrors 281
 Modulo operator 202–3
 Mosaic 312–14
 Motion detection 294–7
 Mouse, variation with 34–8
 Mouse clicks 38–40
 MouseDragged() function 366
 MousePressed() function 39, 66, 67, 68, 72, 93, 340
 Move() function 109, 179, 193
 Movie object 279–81
 MovieMaker 401–3
 MP3 fi les 386
 Multiple rollovers 69–70
 Multi-user communication 362
 client 366–7
 server 362–5
 server and client 368
 MyVar, error messages of 438–9
 NetEvent() function 340, 341, 345
 New PImage() 254
 Newline character 355–6
 NewTab option 128, 129
 NoFill() function 11, 232
 Noise() function 208
 NoLoop() function 114
 NoSmooth() function 27
 NoStroke() function 11
 NullPointerException error 127, 440–1
 Object-oriented programming (OOP) 121, 407
 encapsulation 407–9
 inheritance 410–13
 overloading 419
 polymorphism 416–18
 shapes 413–16
 Object-oriented Zoog 135–7
 Objects 122
 arrays of 152–4
 constructor arguments 131–4
 cookie cutter, writing 124–5
 data types 135
 object-oriented Zoog 135–7
 with OOP 121–2
 usage 122–4
 details 125–7
 with tab 127–31

One-dimensional Perlin noise 208
 OPENGL 230–1
 Origin, definition of 227
 OSC (open sound control) 379
 Oscillation 214–16
 Overloading 419
 P3D 230–1
 PApplet 423
 Particle system 427
 Pass by reference 135
 Pass by value 113
 Passing parameters 111
 Pde file 20, 128, 131
 Perlin noise 207–10
 PFont.list() 308
 Physics 101, 79–81
 PI, definition of 211
 Pixel point 270
 Pixels 3, 260–2
 color transparency 14–15
 custom color ranges 15–16
 graph paper 3–4
 grayscale color 10–12
 PImage object 265–6
 pixel group processing 267–70
 RGB color 12–14
 simple shapes 5–9
 Play() function 279, 380, 384, 387
 Pointillism 270–1
 Polar coordinates 212
 to Cartesian 213
 Polymorphism 416–18
 PopMatrix() function 241, 243, 244, 245, 246, 249
 Port numbers 354
 Present mode 19
 Primitive data types 113, 253
 Primitive shapes 5
 Primitive values 47
 Println 193–4
 Println() function 22
 PrintMatrix() function 241
 Probability review 204–5
 event probability, in code 205–7
 Procedures see Functions
 Processing filter 256–7
 Processing graphics 253
 Processing library 350
 Processing software 17
 application of 18–20
 coding 20–3, 26–8
 errors 23–4

play button 26
 publishing, as Java applet 28–9
 reference 24–6
 sketchbook 20
 ProSVG 400
 Pseudocode 168
 Pseudo-random numbers 203
 Public/private classes, in Java 423
 Publishing, as Java applet 28–9
 Pushing and popping, matrix 240–7
 PushMatrix() function 241, 243, 244, 245, 246, 248
 Putting 'on the Ritz' 182–8
 Quad() 8, 9
 QuickTime libraries 273
 QuickTime movies 401–3
 Radians, definition of 210–11
 Radians() function 211
 Raindrops 178–82
 Random numbers 203–4
 distribution 204
 Random() function 55, 56, 57, 66, 115, 203
 Randomizer() function 113
 Read() function 275
 ReadRawSource() function 340
 ReadString() function 355
 ReadStringUntil() function 364
 Really Simple Syndication (RSS) 344
 Recorded video 279–81
 Rect() function 24, 87, 104, 227, 230, 429
 Rectangle class 429–32
 Recursion 216–20
 Reference, in processing 24–6
 RequestWeather() function 336
 Reserved words 22
 RestoreMatrix() 243
 Return statement 115
 Return type 114–17
 Reverse() function 157
 RGB color 12–14
 Rollover() function 154, 330
 Rotate() function 210, 235, 236, 238, 256
 RotateZ() function 242, 244
 Rotation, around axes 237–40
 Sandbox 298, 352
 SaveData() function 331
 SaveFrame() 400–1
 SaveMatrix() 243
 SaveString() function 331, 332
 Scale 240

Scribbler mirror 286–7
 Search() function 350 , 351
 SearchEvent() 350
 Serial communication 368
 with handshaking 371–2
 with strings 372–4
 SerialEvent() function 370
 Server:
 creation 354–7
 for multi-user communication 362–5
 ServerEvent() function 355
 SetRate() function 387
 Setters 408
 Setup() functions 32–4 , 38 , 75 , 93 , 103 , 105 , 123 , 150 , 182 , 183 , 242 , 255 , 307
 SetVolume() function 386
 Shapes 413–16
 Shorten() function 157
 Simple background removal 293–4
 Simple color tracking 290–1
 Simple modularity 106–8
 Simple rotation 235–7
 Simple shapes 5–9
 SimpleML library 197 , 339 , 340–2
 SimplePostScript 400
 Sin() function 213
 Sine 212 , 213 , 216
 Size() function 21 , 229 , 230 , 231
 Sketchbook 20
 Sketches 18 , 20
 SketchName.exe 396
 Smooth() function 27
 Socket connection 354
 Software mirrors 281–7
 Sohahtoa 212
 Solar system, processing 247
 object-oriented 248–9
 Sonia library 379 , 380–1
 Sonia.stop() function 381
 Sort() function 157
 Sound 379
 libraries for 380–1
 LiveInput 388–90
 playback 381–6
 volume, pitch, and pan control 386–8
 thresholding 390–3
 Sound events, with Sonia 392–3
 Source directory 396
 Spatial convolution 268
 Splice() function 157
 Split() function 325 , 326
 Stand-alone applications 395–7
 Start() function 183
 State variable 78
 Static functions 389
 Strings 303–6
 manipulation 323–4
 serial communication with 372–4
 Stripe object 154–5
 Stroke() function 10 , 11 , 24 , 195
 Subroutines see Functions
 Subset() function 157
 Substring() function 323 , 324 , 334
 Super() 412 , 414
 SVG (Scalable Vector Graphics) 400
 Synchronous vs. asynchronous requests 353–4
 System variables 54–5
 Tangent 212
 Telnet client 357
 Text 303
 analysis 338–9
 animation 309–12
 character by character, displaying 315–21
 displaying 306–9
 mosaic 312–14
 parsing 333–8
 rotating 314–15
 strings 303–6
 Text files, reading and writing of 327–33
 creating objects 329
 loading and saving data 331–3
 Text() function 316
 TextAlign() function 309
 TextFont() function 308
 TextLeading() function 311
 TextMode() function 311
 TextSize() function 311
 TextWidth() 310 , 311 , 316
 Th estring.length() 324
 “Th ing” class, error messages of 442
 Third party library 197
 Threshold filter 265
 Timer 176–8
 Tint() function 256 , 257 , 264–5
 ToUpperCase() function 323
 Transformation matrix 241
 Translate() function 227 , 228 , 229 , 230 , 236 , 244 , 256
 Translation and rotation, in3D 225
 custom 3D shapes 233–5
 different axes, of rotation 237–40
 OPENGL 230–1
 P3D 230–1
 pushing and popping 240–7
 scale 240
 simple rotation 235–7

solar system, processing 247–9
vertex shapes 231–3
z-axis 225–30
Triangle() 8 , 9
Trigonometry 212–14
Trim() function 356
Try catch 432
Two-dimensional arrays 220–4
of objects 223–4
Type, defi nition of 47
UpdatePixels() function 261 , 292
UpperCase() method 305
USB (Universal Serial Bus) 369
User defined functions 104
Variable names, tips for 48–9
Variable scope 93–5
Variable Zoog 57–9
Variables 45
declaration and initialization 47–9
many variables 52–3
system variables 54–5
usage 49–52
variable Zoog 57–9
variety, spice of life 55–7
Variety, spice of life 55–7
Vertex() function 231 , 233
Vertex shapes 231–3
Video 273
before processing 273
background removal 292–4
computer vision 287–91

libraries 297–8
image manipulation 277–8
live video 101 , 274–8
motion detection 294–7
pixelation 283–4
recorded video 279–81
sandbox 298
as sensor 287–91
software mirrors 281–7
Visualization 270–2
Volume, pitch, and pan control 386–8
WAV (Waveform audio format) 380
Web applets 395
“ While ” loop 85–8
Whole numbers 47
Wiring web sites 369
Withdraw() function 408
XML 335 , 342–6
XML library, processing 346–9
XMLElement object 347
XMLRequest() 345
Yahoo API 349–52
Z-axis, in 3D space 225–30
Zoog 16 , 97–9
in arrays 159–60
as dynamic sketch 34
with variation 36
OOP 135–7
reorganization 118–19