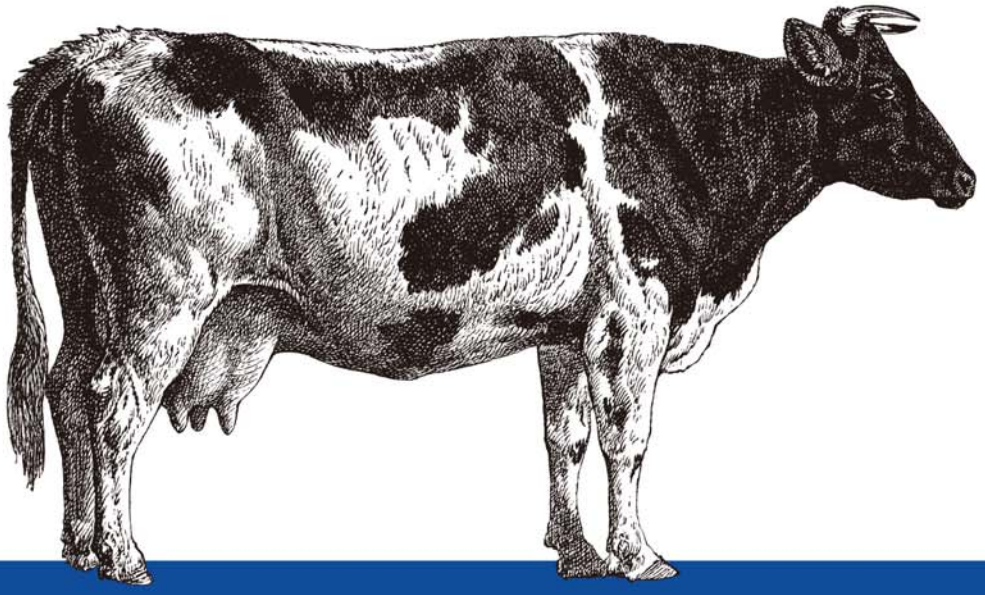
O'REILLY®



奔跑吧 Ansible

Ansible: Up and Running 让自动化配置管理与部署变得简单

[加] Lorin Hochstein 著
陈尔冬 译

中国工信出版集团
電子工業出版社 PUBLISHING HOUSE OF ELECTRONICS INDUSTRY http://www.phei.com.cn

O'REILLY®

奔跑吧Ansible

Ansible: Up and Running

[加] Lorin Hochstein 著

陈尔冬 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

Ansible是近年用户量急速蹿升的开源配置管理工具。在Ansible之前，行业中已经有很多开源配置管理工具了，特别是鼎鼎大名的Puppet，堪称配置管理界的超级巨星。然而，Ansible依靠它的简单易用、“零依赖”及弱抽象还是获得了无数开发者和运维工程师的青睐。遗憾的是，由于Ansible还很年轻，除了官方文档外，其他相关的优秀文档可谓凤毛麟角。而本书恰恰就是为了缓解这一现状而写的。作者在本书中演示了如何使用Ansible在接近真实的生产环境进行管理的案例，这既演示了Ansible的强大功能，又能够帮助读者快速入门与上手，非常适合作为官方文档的扩展资料来阅读。

版权贸易合同登记号 图字：01-2015-5385

图书在版编目（CIP）数据

奔跑吧Ansible /（加）霍克斯坦（Hochstein,L.）著；陈尔冬译. —北京：电子工业出版社，2016.1
书名原文：Ansible: Up and Running
ISBN 978-7-121-27507-4

Ⅰ. ①奔… Ⅱ. ①霍… ②陈… Ⅲ. ①程序开发工具 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2015)第263674号

策划编辑：张春雨
责任编辑：付 睿
封面设计：Ellie Volkhausen 张 健
印 刷：北京天宇星印刷厂
装 订：北京天宇星印刷厂
出版发行：电子工业出版社
北京市海淀区万寿路173信箱 邮编：100036
开 本：787×980 1/16 印张：21.75 字数：476千字
版 次：2016年1月第1版
印 次：2016年1月第1次印刷
定 价：79.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

O'Reilly Media, Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar 博客有口皆碑。”

——Wired

“O'Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去 Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

推荐语

大规模集群的配置管理工具一直都是提升效率的利器。新浪在 2003 年前后开始使用 Cfengine；在 2010 年前后，为了解决 Cfengine 的一些问题，我们逐渐在一些业务中用 SSH 进行批量命令执行和配置文件拉取，这种组合一直用到现在，管理着数万台服务器，但现在基本上已经是 SSH 为主了。本文的译者在新浪负责了很多配置管理系统的开发，以我们的经验来看，基于 SSH 进行配置管理有很多好处，这也是我对 Ansible 很感兴趣的地方。充分利用系统现成的机制，这不仅省去了 Agent 的安装，在数千台规模的大型数据中心，少一些中心化服务还意味着少了一些路由和安全策略的配置，省去了很多不必要的麻烦。当然，Ansible 还有很多吸引人的地方，非常值得你去学习和发现。

——童剑

新浪研发中心总经理

Ansible 作为自动化系统运维的一大利器，在构建整个体系过程中有着举足轻重的地位。DSL、幂等性、playbook、大量的模板等都是它的魅力所在，再加上易封装、接口调用方便，Ansible 正在被越来越多的大公司采用，包括独立使用以及与其他工具（如 Puppet）结合使用。

本书通过简单易用的大量实例帮忙读者快速上手，通读全书会让你对 Ansible 有一个全面的了解，动手操作书中的实例，快速掌握 Ansible，剩下的就交给你的灵感吧。

——刘宇

@ 守住每一天，金山西山居架构师

《Puppet 实战》作者，《Puppet 实战手册》译者之一

对于 DevOps，我自己的理解是运维即开发，人管理代码，代码管理机器，而不是人直接管理机器。Ansible 帮我们实现了运维人员开始向运维开发的转型，让繁杂、危险的运维工作变得简单、安全和可控。《奔跑吧 Ansible》秉承了 Ansible 这一工具简明的一贯特点，不是长篇累牍地讲解复杂技术原理，而是列举了大量简明的实例，拿来阅读半小时即可上手解决实际问题。

——程辉

UnitedStack 公司创始人兼 CEO

推荐序一

几年前，我和尔冬都作为技术团队的一员在一起开始新浪微博的研发，研发团队开发完第一个服务后就面临服务发布及管理的问题。最开始是通过工程师手动登录服务器的方式来发布与启动。当服务请求增大、服务单元增多后，工程师便将命令写成脚本，通过SSH在多台机器执行，半人工地解决了一段时间的问题。当服务规模进一步增大，手工运行SSH变得困难后，我们开始寻找一些工具来使这些工作自动化，当时能找到的工具都需要在机器上安装一个agent，通过agent接收指令来执行对应的操作。随着服务集群规模变大、依赖变多，当时使用的工具需要具备指令安全、执行进度汇报、执行结果检测等功能。于是我们自己做了一个脚本，可以通过命令行的方式控制软件发布、服务启动与进度查看。随着机器规模的增加，命令行及脚本的方式显得不是很直观，也不利于新进入团队的人了解及使用。于是我们做了一个Web界面来中心化管理及执行脚本，并且可以很直观地查看进度，还可以选择发布任务执行的范围，较好地解决了当时服务发布遇到的问题。

几年之后，我们发现Ansible使用相同的理念更好地解决了以上问题。其设计方法也遵循“脚本+可视化管理”的思路，Ansible本身是一种脚本控制的语言，在此之上，我们可以选择其商业化的Tower软件来可视化地进行管理。而其脚本在指令安全、执行进度汇报、执行结果检测方面，相对于直接运行操作系统脚本都具有更大的优势。

自动化管理是大势所趋，尔冬及时翻译的《奔跑吧Ansible》一书可以给国内同行带来很多启发，希望大家能够利用Ansible工具及设计思想将应用的发布和配置管理提升到新的高度。

——TimYang

微博研发副总经理，“高可用架构”公号主

推荐序二

运维发展到今天，已经不是刀耕火种的年代，各种运维自动化工具层出不穷，运维人员也逐步摆脱了直接登录服务器进行操作所带来的烦琐、重复和高风险性。自动化工具也有一个演进过程，从 Puppet、Saltstack 到 Chef，运维人员在学习和使用的过程中也深深因其复杂性而苦恼，特别是客户端机制。相比之下，Ansible 的使用简单很多，这也是它广受欢迎的主要原因。

运维自动化工具本来是用来简化运维工作的，但如果工具本身比较复杂，甚至需要一定的程序开发能力，就会增加使用和推广的难度。Ansible 有三个最吸引人的地方：无客户端、简单易用和日志集中存储。很多时候，运维人员对服务器仅具有临时权限，甚至没有权限，所以无法部署客户端程序。另外，客户端机制往往也是运维自动化容易出问题的地方，这涉及客户端的安装、配置修改和卸载，其中任何一步没有同步完成，都可能会带来隐患。Ansible 很简单，上手方便，不需要啃一本大部头书才能学会使用（从这一点来看，真可谓业界良心）。另外 Ansible 又很好地解决了 Shell 操作日志的集中存储问题——这一点在被管理服务器数量少时，貌似作用还不大，但在批量管理大量服务器时，就能显示出其便利性了。所有操作日志都存储在 Ansible 发起服务器，可以采用自定义的格式，这样可以很方便地知晓哪些服务器的操作有问题，哪些已成功，而且便于日后的追溯。当然，Ansible 的配置管理功能简单而强大，所有被管理的软件配置都集中存储，如果目标服务器想安装 MySQL，InnoDB 所占用内存需要从 10GB 调整为 12GB，那么在 Ansible 发起服务器简单修改即可。

我对 Ansible 的印象非常好。在 2014 年年底的时候，为了给公司采购公有云提供决策支持，我选定了国内八大公有云，对它们进行了长达四个月的测试，总共进行了上万次测试。每家公有云随机选择 3~5 台云主机和 RDS，每台云主机测试多轮，每轮测试 200 次。测试项目包括网络稳定性（探测节点涵盖全国共 400 多个）、CPU、内存、磁盘及云主机

整机性能测试。如果不借助于 Ansible，仅仅两个人是无论如何也不可能在短时间内完成一万多次测试的。我们基于 Ansible 编写了一个批量自动化测试工具，这个工具完成了对新入手云主机的一切工作，包括初始化、测试工具部署、测试数据装载及自动发起指定次数的测试流程。结果很完美，所有测试结果都汇聚在 Ansible 发起服务器，我们又开发了一个日志自动分析的工具，能从 Ansible log 中截取有用的信息并加以汇总（就只差用 Excel 出图了）。

同时，Ansible 虽然不如 Puppet 等复杂，但也还是需要一些方法和技巧，而且版本更新迭代较快。因此我也很高兴看到尔冬兄亲自翻译的《奔跑吧 Ansible》这本书，其英文原版由 Michael DeHaan（Ansible 软件的创作者，Ansible 公司前 CTO）亲自作序，其受认可度可见一斑。本书内容编排由浅入深，理论与实践并重，作者特别提到了 SSH multiplexing，Ansible 与之配合，可以用来管理成千上万的服务器节点。

尔冬兄是运维行业资深人士，深度参与了新浪微博从小到大的发展过程。每次和尔冬兄交流，总能感觉到他对运维行业深深的感情，以及关于运维的真知灼见。这次尔冬兄亲自翻译此书，可以说是国内诸多 Ansible 使用者的福祉，可以帮助大家更好地学习、理解、掌握 Ansible，并融会贯通。

——萧田国

开放运维联盟联合主席，高效运维社区发起人

译者序

由于诸多原因，早在童年时代计算机就进入了我的生活。对计算机的迷恋最终将我带进了计算机工程领域。而让我真正从玩耍转向工程化地对待计算机的分水岭就是，对系统管理领域的接触。好吧，不管多么不乐意，这类工作在那个年代就是被称作网管。从我刚开始入门系统管理的时候，我就一直有一个疑问：如果一家公司足够大，有上百台计算机，该怎么管呢？总不能一个个远程桌面连上去吧？嗯，没错，那个时候我使用的还是 Windows，而上百台计算机对于当时的我来说已经是一个很夸张的数字了。

2007 年我加入了新浪平台架构部，这里的工作为我真正打开了新世界的大门。那个时候部门正在使用 CFEngine 管理上百台机器、为不同功能的服务器划分角色、为相同功能的服务器进行编号（像为公牛编号那样）、为配置文件编写模板来减少硬编码，所有这一切都用一种工程实践的方法解决了我之前的所有疑问。后来，我才知道这个实践性很强的方法的名字——配置管理。

转眼之间，我已经在新浪工作了七年之久。这七年间我的职位与工作内容有多次变化，但所围绕的工作核心从未改变：如何让数千台服务器按照我们想要的方式运转。为了更好地达到这一目的，我尝试过各种配置管理的方式：从 CFEngine 到 Puppet、SaltStack，甚至是自行开发配置管理工具。但是每一种方式都与我理想中那个遵循“KISS 原则”、易于学习，且在功能上具有无穷扩展空间的配置管理工具相差甚远——直到经同事文旭的推荐，我认识了 Ansible。Ansible 的轻量、最小化抽象层及轻松扩展与收缩一下子就吸引了我。目前为止，它是与我理想中的配置管理工具最接近的一个。

这样优秀的工具我当然不会自己独享。我曾经在各种场合向正在寻找合适配置管理工具的朋友与同事推荐 Ansible。我发现有一部分朋友虽然对于 Ansible 给予了正面的评价，却对缺少中文文档感觉略有不便。这让我意识到语言仍旧是部分技术人员学习技术的障碍之一。显然，我并不具备帮助技术人员提升英语阅读水平的能力，但至少我可以将这

本《奔跑吧 Ansible》的中文版带给大家，希望本书可以帮助一些读者快速上手 Ansible。

由于水平所限，本书中难免出现一些翻译错误。诚恳地欢迎大家向我或者出版社反馈本书中的各种错误。

最后，我想要感谢赵新宇、陈明杰和刘宇等朋友，他们在翻译本书过程中提供了无私的帮助和支持。还要感谢我的夫人张若金的支持与理解。没有你们就不会有本书的出版。

目录

原书推荐序

在 2012 年 2 月创立的时候，Ansible 还是一个非常简单的项目，随后它的快速发展令我们倍感惊喜。现在，它已经是上千人参与开发的产品了（如果包括参与贡献想法的人，还会更多），并且广泛部署于几乎每个国家。在各种技术会议中总是能找到有（至少）几个人在使用它，这在计算机领域也是件很不寻常的事。

Ansible 的不平凡源自于它的平凡。Ansible 并不企图做盘古开天地般的创新，而是从那些聪明的家伙们已经提出的想法中提炼出精华，并将这些想法尽可能地落地。

Ansible 旨在探求某些学术的 IT 自动化方法（它们本身就是对大型繁杂的商业套件的一种反应）与简单粗暴解决问题的脚本之间的平衡点，另外，我们如何能将配置管理系统、部署发布系统、编配系统（orchestration project）以及千奇百怪但是非常重要的 Shell 脚本库替换为一个单一系统呢？这恰恰是 Ansible 要实现的理念。

我们可以从 IT 自动化技术栈中移除主要架构组件吗？去掉管理性守护进程，转而依赖于 OpenSSH，意味着系统转眼间就可以开始管理一台新的计算机，而不需要在被管理的机器上安装任何东西。更深一层来说，系统更趋于可靠和安全了。

我注意到，提前尝试使用自动化系统本该使事情变得简单，但实际上却变得更难了。并且编写以自动化在以前为目的的东西就好像个吸收时间的黑洞，使我无法在本应该更专注的事情上投入更多时间。况且我并不想在这种系统上投入数个月以成为这个领域的专家。

我个人尤其享受编写新的软件，而不喜欢在使其自动化方面花太多时间。简而言之，我希望自动化的事情尽快完成，这样我就能有更多时间投入在我更关注的事情上面。Ansible 并不是一个你需要整天和它打交道的系统。你可以很快把它拿起来，很快搞定，然后又很快回到你更关心的事情上面。

我希望这些也会成为你喜欢 Ansible 的原因。

尽管我花了大量时间来确保 Ansible 的文档易于理解和掌握，但是有不同形式的材料可以参考，并依此尝试实践应用总是大有裨益的。

在《奔跑吧 Ansible》一书中，Lorin 使用非常流畅的行文、适于逐步探索的顺序介绍了 Ansible。Lorin 几乎是从最开始就参与到了 Ansible 项目中，我真诚地感谢他做出的贡献。

我还要真诚地感谢今天项目中的每一位成员，以及未来的每一位成员。

最后，希望你们喜欢这本书，享受瞬间就可以管理你的计算机的愉悦感！啊，对了，别忘记安装 cowsay[注 1]！

——Michael DeHaan

Ansible 软件的创作者，Ansible, Inc. 公司前 CTO[注 2]

2015 年 4 月

注 1：Ansible 的创作者很有趣。从 0.5 版本开始他给 Ansible 留了个彩蛋：如果你的机器安装了 cowsay 的话，执行 playbook 的时候，终端上就会显示一只奶牛。最主要的是，他还在发布 0.5 版本的时候严肃地介绍了这么做的优点。——译者注

注 2：如 Michael DeHaan 自己所说，他其实更享受编写新软件。目前他已经从 Ansible 公司离职到 DataStax 工作。在 Ansible 公司的最后工作日，他写下了一篇博文：*Happy Trails, Ansible*（*http://michaeldehaan.net/post/109595670406/happy-trails-ansible*）。——译者注

前言

我为什么写这本书

多年前，我使用时下流行的 Python Web 框架 Django 编写了我人生中的第一个 Web 应用，当那个应用终于在我的台式机成功运行时，油然而生的成就感让我至今难忘。首先运行 django manage.py runserver，然后打开浏览器并访问 http://localhost:8000，最后是见证奇迹的时刻——我的 Web 应用闪亮登场！

然而，随后我就发现让该死的应用运行在 Linux 服务器上其实有非常多令人无奈的事情。除了把 Django 本身和我的应用安装在服务器之外，我还必须得安装 Apache 和 `mod_python` 模块。有了 mod_python 模块，Apache 才可以运行 Django 应用。这还不算完，我还必须弄明白天书一般的 Apache 配置文件，并把它配置为可以同时运行我的程序和其他所依赖的静态内容。

说实在的，每一步都不算难，但是想让所有的环节都配置正确也是个痛苦的过程。作为一名程序员，我才不想不停地摆弄配置文件，我只是想让我的应用运行而已。好在当我把一切都配好了之后，就不用再去动了。然而几个月后的一天，噩耗传来，在另一台服务器上还得再经历这些无奈，而且还是从头开始。

终于有一天，我发现痛苦的根源在于我用的方法不对。做这些事情的正确方法有个学名：*配置管理*。使用配置管理要注意的一个重要的事情是：它是一种获取那些始终保持最新的配置与信息的方法。你不再需要频繁地搜索正确的文档或者查找以前的笔记。

不久以前，一位同事出于兴趣尝试使用 Ansible 部署新项目，他请我帮忙推荐一些官方文档以外的 Ansible 应用实践方面的参考资料。那时我才突然发现，好像官方文档以外没有什么资料可以推荐的了。于是我决定填补这个空白，所以就有了这本书。可惜这本

书对于那位同事来说太迟了，但希望对于你来说它来得正是时候。

谁适合读这本书

这本书是写给需要管理 Linux 或者类 Unix 服务器的人的。如果你对下列术语如数家珍：系统管理、运维、部署、配置管理（看到这也许有的读者在叹气）或者 *DevOps*，那么看到本书你也许会觉得如获至宝。

尽管我也在管理自己的 Linux 服务器，但我的专业领域其实是软件开发。这意味着本书的范例将偏向于部署领域，尽管我同意 Andrew Clay Shafer（见附录 D 参考文献 webops）的部署与配置没有明确边界的观点。

本书引导

我对这样的提纲并不感冒："第 1 章涵盖这个；第 2 章涵盖那个"，类似这样的内容不会有人真正仔细去看（反正我从没看过）。我认为目录反倒更易于浏览。

本书的内容编排是适于从前向后顺序阅读的，后面的章节会基于前面的内容。本书包含大量范例，建议你按照本书指导在自己的计算机上去试验这些范例。绝大部分范例都是面向 Web 应用的。

本书约定

本书使用的排版约定如下：

斜体（*Italic*）
: 新术语、URL、电子邮件地址、文件名和文件扩展名以这种字体展示。

等宽字体（`Constant width`）
: 程序代码或正文中包含的程序元素，如变量、函数名、数据库、数据类型、环境变量、语句和关键字等代码文本以这种字体展示。

等宽黑体字（**`Constant width bold`**）
: 需要用户输入的命令或其他文本以这种字体展示。

等宽斜体字（*`Constant width italic`*）
: 需要由用户替换为具体内容的文本，或需要遵照具体上下文确定具体内容的文本由这种字体展示。

这个图标表示小窍门、建议或是一般注意事项。

这个图标表示一般性的提示。

这个图标表示警告或提醒。

本书切口处 ⇨ 中的页码对应英文原书页码。

在线资源

本书中的样例代码都可以在本书的 GitHub 页面（*http://github.com/lorin/ansiblebook*）上获取到。Ansible 官方也提供了丰富的文档（*http://docs.ansible.com*）供参考。

我还在 GitHub 上维护了一个简明参考手册（*http://github.com/lorin/ansible-quickref*）。

Ansible 的代码存放在 GitHub 上，分割成三部分分别存放在不同的代码仓库上：

- 主仓库（*https://github.com/ansible/ansible*）。
- 核心模块（*https://github.com/ansible/ansible-modules-core*）。
- 其他模块（*https://github.com/ansible/ansible-modules-extras*）。

建议将 Ansible 的模块索引加到你的收藏夹中，在使用 Ansible 过程中，你将会时常需要去查看它，网址为：*http://docs.ansible.com/modules_by_category.html*。Ansible Galaxy 是一个由社区共同维护的 Ansible 角色仓库，网址为：*https://galaxy.ansible.com/*。

如果你有任何关于 Ansible 的问题，都可以到 Ansible 项目的 Google Group 中讨论：*https://groups.google.com/forum/#!forum/ansible-project*。

如果你希望向 Ansible 开发组贡献代码，可以访问 Ansible 开发组的 Google Group：*https://groups.google.com/forum/#!topic/ansible-devel/68x4MzXePC4*。

在 *irc.freenode.net* 上还有个非常活跃的 *#ansible IRC* 频道，在那里你可以得到更为实时的帮助。

本书旨在帮你解决实际问题。一般来说，除大批量使用代码之外，如果你需要在自己的程序或文档中使用本书提供的范例代码是不需要联系我们取得授权的。例如，使用本书的几段代码编写一个程序不需要向我们申请许可。但是销售或者分发 O'Reilly 图书随附的代码光盘则必须事先获得授权。引用书中的代码来回答问题也无须我们授权。将大段的示例代码整合到自己的产品文档中则必须经过我们的授权。

我们非常希望你能在引用本书时表明出处，但并不强求。出处一般包含有书名、作者、出版商和 ISBN。例如："*Ansible: Up and Running* by Lorin Hochstein (O'Reilly). Copyright 2015 Lorin Hochstein, 978-1-491-91532-5"。

如果有关于使用代码的未尽事宜，可以随时与我们联系：*permissions@oreilly.com*。

Safari Books Online

Safari Books Online 是应需而变的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。

技术专家、软件开发者、Web 设计师、商务人士和创意精英都可以将 Safari 在线图书作为他们的调研、解决问题、学习和认证的主要资料来源。

Safari Books Online 对于组织团体、政府机构和个人提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sam、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGarw-Hill、Jones & Bartlett、Course Technology 及其他数十家出版社的上千种图书、培训视频和正式出版前的书稿。要了解更多关于 Safari Books Online 的信息，请访问我们的网站。

联系我们

请将对本书的评价和发现的问题通过如下地址告知出版者。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

本书拥有一个专属网页，你可以在那找到本书的勘误表、范例代码和其他信息，网址为：*http:// bit.ly/ansible-up-and-ruuning*。

对本书的意见和技术性问题还可以发送电子邮件到 *bookquestions@oreilly.com*。

想要了解更多关于 O'Reilly 图书、培训课程、会议和新闻等信息，请访问 O'Reilly 官方网站：*http://www.oreilly.com*。

Facebook 地址：*http://facebook.com/oreilly*。

Twitter 地址：*http://twitter.com/oreillymedia*。

YouTube 地址：*http://www.youtube.com/oreillymedia*。

致谢

感谢 Jan-Piet Mens、Matt Jaynes 和 John Jarvis 审阅本书的草稿并提出反馈意见。感谢在 SendGrid 工作的 Isaac Saldana 和 Mike Rowan 的大力支持。感谢 Michael Dehaan 创建了 Ansible 项目并引领着迅速崛起的 Ansible 社区。同时也感谢 Michael Dehaan 对本书的反馈，包括他对 Ansible 名字由来的讲解。感谢本书的编辑 Brian Anderson 超耐心地与我共同工作。

感谢我父母无私的支持；感谢我的哥哥 Eric，他是我们家族中的正牌作家；感谢我的两个儿子 Benjamin 和 Julian；最后，感谢我的妻子 Stacy 在各方面的付出与无条件的支持。

第 1 章

概述

在 IT 行业工作很有意思。向客户交付软件的时候，“只要在一台机器上装好软件就能搞定收工注 1”这种事情只能存在于理想中。实际上，我们都慢慢变成了系统工程师。

我们现在部署应用时需要把不同的服务组合起来。这些服务运行在一组分布式计算资源上，并且使用不同的网络协议相互通信。一个应用一般由 Web 服务、应用服务、内存缓存系统、任务队列、消息队列、SQL 数据库、NoSQL 数据库及负载均衡等几部分组成。

我们还得保证服务具有适当的冗余，这样出现异常（放心，异常一定会有的）时，我们的软件可以平滑地处理这些异常。配套辅助也是我们自己去部署和维护的。辅助服务一般包括日志系统、监控系统和数据分析系统。有时候辅助系统还包括我们所依赖的第三方服务，比如用于管理虚拟机实例注 2 的 IaaS 终端。

你完全可以手工配置这些服务：SSH 登录到一台服务器，安装软件包，编辑配置文件，然后换下一台继续。想想就痛苦！特别是在重复到三四次的时候，你会感到它出奇地费时费力、枯燥乏味，还特别容易出错。而且还有很多更复杂的任务，比如在你的应用中搭建 OpenStack 云，全手动来完成是个很疯狂的想法。幸运的是，我们有更好的方式。

既然你在读这本书，那么你应该热衷于配置管理理念，并且在考虑采用 Ansible 作为你的配置管理工具。不管你是准备向生产环境部署自己代码的开发者，还是寻求更好的自动化解决方案的系统管理员，我相信你都会觉得 Ansible 是解决你问题的完美方案。

注 1：　好吧，其实软件部署从来没这么简单过。

注 2：　可以参考 *The Practice of Cloud System Administration* 和 *Designing Data-Intensive Application*。这两本超棒的书就是描述构建和维护这种类型的分布式系统的。

关于版本的说明

本书的所有范例代码都在 Ansible 1.8.4 版本上经过了测试，这是写作本书时的最新版本。由于 Ansible 项目的主要目标之一就是向下兼容，所以这些范例应该不经修改就可以在之后的版本上正常运行。

溯源“Ansible”

Ansible 出自科幻小说，是虚构的一种以超光速传递信息的通讯装置。Ursula K. Le Guin 在他的 *Rocannon's World* 一书中提出了这个概念。后来，有很多其他的科幻作家借鉴过这个概念。

实际上，Ansible 的作者 Michael DeHaan 是从 Orson Scott Card 撰写的 *Ender's Game* 一书中借鉴来的。在这本书中，Ansible 可以跨越任何距离同时控制无数飞船，就好像在我们的世界控制海量远程服务器一样。

Ansible 的优点

Ansible 被定义为配置管理工具，并且通常与 *Chef*、*Puppet* 及 *Salt* 相提并论。当我们提到配置管理的时候，通常都会联想到编写一个描述所有服务器状态的配置文件，并使用工具确保所有服务器都保持在那个状态之上。这些状态包括但不限于：确保所依赖的软件包已经被安装、配置文件包含正确的内容和正确的权限、相关服务被正确运行，等等。Ansible 引入一种特定领域语言（domain-specific language，缩写 DSL），你可以使用 DSL 来描述服务器的状态。

其实这些工具也可以用于部署。通常意义的部署指的是将自研软件编译成二进制文件、生成相关的静态资源、将所有必需的文件复制到服务器上并将服务启动起来这一完整过程。*Capistrano* 和 *Fabric* 就是两个开源部署工具的例子。Ansible 不仅是配置管理的利器，
3 也是用于部署的神兵。对于负责运维的人来说，用一个简单的工具就可以同时完成配置管理和部署的工作，生活都瞬间变得美好了。

有些人会讨论部署中对编配（Orchestration）的需求。他们指的是如何解决各种远程服务器正确提供服务，以及保证各种操作以特定的顺序执行的问题。例如，你需要在启动 Web 服务器之前先启动数据库，或者为了实现零停机升级，你需要将 Web 服务器逐一从负载均衡上摘除并升级。实际上，这也是 Ansible 擅长的，Ansible 的设计初衷就是在若干服务器上从零开始执行所有必需的配置与操作。Ansible 利用极度简洁的模型来控制各种操作按照所需顺序执行。

最后，我们来聊聊新服务器的置备（provisioning）。在 Amazon EC2 这种公有云环境，置备就是指创建新的虚拟机实例。Ansible 也能覆盖这个领域。Ansible 具有大量模块可以和 EC2、Azure、Digital Ocean、Google Compute Engine、Linode、Rackspace 及其他支持 OpenStack API 的公有云通信。

本章马上会讨论到的工具 *Vagrant* 有一个特别容易引起混淆的地方，它使用术语“置备器（provisioner）”来指代用于配置管理的工具。因此，Vagrant 将 Ansible 看作一种置备器。而我认为 Vagrant 才是置备器，因为 Vagrant 负责创建虚拟机。

Ansible 如何运作

图 1-1 展示了一个简单的 Ansible 使用案例。有一位名为 Stacy 的用户使用 Ansible 来管理三台基于 Ubuntu 发行版并且运行 Nginx 的 Web 服务器。她编写了一个 Ansible 脚本：*webservers.yml*。Ansible 中将脚本（script）称作 *playbook*。playbook 描述了哪些主机（Ansible 将其称为远程服务器）需要配置，以及需要在那些主机上运行的有序任务列表。在这个范例中，主机就是 Web1、Web2 和 Web3，而任务则包括：

- 安装 nginx。
- 生成 nginx 配置文件。
- 复制 SSL 证书到相应位置。
- 启动 nginx 服务。

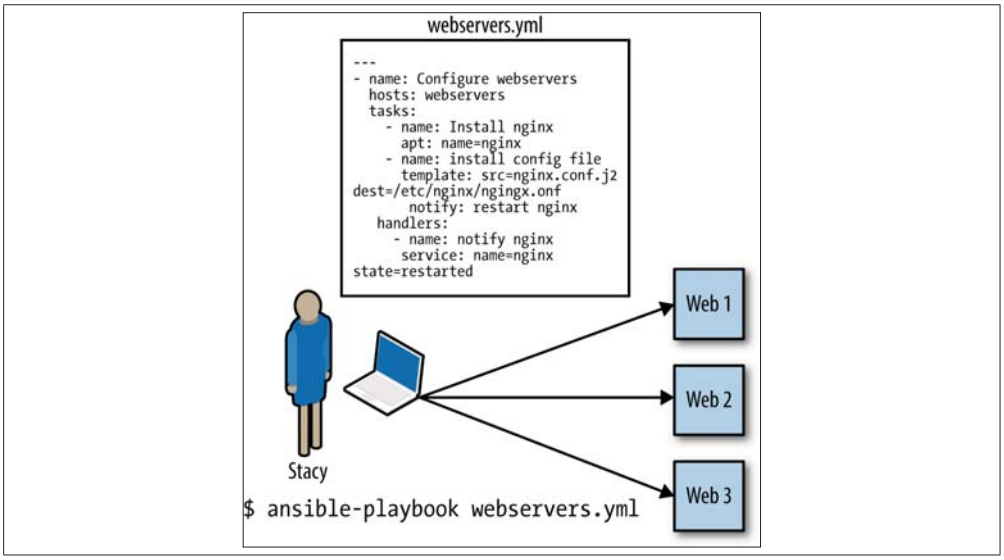


图1-1　运行一个Ansible playbook来配置三台Web服务器

我们将在下一章详细讨论 playbook。Stacy 使用命令 *ansible-playbook* 来执行 playbook。在本范例中，playbook 的名字是 *webservers.yml*，所以用以下命令执行：

```
$ ansible-playbook webservers.yml
```

4 Ansible 将会并行地建立连接到 Web1、Web2 和 Web3 的 SSH。接着，将会同时在三台主机上执行任务列表中的第一个任务。在这个范例中，第一个任务是安装 nginx 的 apt 软件包（因为 Ubuntu 使用 apt 作为包管理工具）。该任务在 playbook 中的描述如下：

```
- name: install nginx
  apt: name=nginx
```

根据上面的配置，Ansible 将会做如下操作：

1. 生成安装 nginx 软件包的 Python 程序。
2. 将此程序复制到 Web1、Web2 和 Web3。
3. 在 Web1、Web2 和 Web3 上执行此程序。
4. 等候该程序在所有主机上完成。

之后，Ansible 将会按照相同的步骤继续进行任务列表中的下一个任务。需要格外注意的是：

- 对于每一个任务，Ansible 都是在所有主机之间并行执行的。
- 在开始下一个任务之前，Ansible 会等待所有主机都完成上一个任务。
- Ansible 会按照你指定的顺序来运行任务。

5 Ansible 的精妙设计有哪些

现在有很多流行的开源配置管理工具可供选择。下面我们会聊一聊我偏爱 Ansible 的原因。

易读的语法

前面我们提到 Ansible 的配置管理脚本叫 *playbook*。Ansible 的 playbook 语法是基于 YAML 开发的。而 YAML 是一种以易于人类读写为设计理念的数据格式语言。某种程度上说，YAML 之于 JSON 就好像 Markdown 之于 HTML。

我喜欢把 Ansible 的 playbook 看作可执行的文档。这就像是一个 README 文件，它里面描述了那些为了部署你的软件所使用的命令。只不过这个命令永远不会因为忘记更新

而过期，因为它们就是直接执行的代码。

远程主机无须安装任何依赖

使用 Ansible 管理的服务器需要安装 ssh 和 Python 2.5（或更新版本），或者安装 *simplejson* 库的 Python 2.4。除此以外，不再需要预装任何 agent 程序或其他软件了。

控制主机（用于控制远程主机的那台主机）需要安装 Python 2.6 或者更高版本。

有些模块也许依赖 Python 2.5 或更新版本，也有些模块可能会需要额外的依赖。请查阅相应模块的官方文档来确认具体依赖信息。

基于推送模式

以 Chef 和 Puppet 为代表的使用 agent 程序的配置管理系统默认使用“拉取模式”。安装在服务器上的 agent 程序定期向中心服务报备状态并拉取相应的配置信息。这种场景下，要让配置变更在服务器上生效需要如下步骤。

1. 你：对配置管理脚本进行变更。
2. 你：将变更内容更新到配置管理中心服务。
3. 服务器上的 agent 程序：当周期计时器到期时被唤醒。
4. 服务器上的 agent 程序：连接到配置管理中心服务。
5. 服务器上的 agent 程序：下载新的配置管理脚本。
6. 服务器上的 agent 程序：在本地执行那些改变服务器状态的配置管理脚本。

6

作为对比，我们来看一下 Ansible 默认采取的推送模式所需的变更步骤。

1. 你：在 playbook 中进行变更。
2. 你：运行新的 playbook。
3. Ansible：连接到服务器并执行那些改变服务器状态的模块

一旦你运行 `ansible-playbook` 命令，Ansible 马上连接到远程服务器并开始工作。

基于推送模式的方式最显著的优点是：直接由你来控制变更在服务器上发生的时间。你

不需要等着计时器过期。拉取模式的拥趸号称拉取模式在大规模服务器场景下具有较好的扩展性，并且更适合处理新服务器随时上下线的场景。然而，我们将在本书后面章节讨论到，Ansible 已经成功地在具有成千上万个节点的生产环境中工作，并且对于服务器动态上下线的场景可以完美支持。

如果你真的更喜欢拉取模式，Ansible 官方版本也可以支持。你可以使用一个名为 ansible-pull 的工具，它是在 Ansible 官方版本内一起发布的。我在本书中不会介绍拉取模式，你可以在官方文档（*http://docs.ansible.com/playbook_intro.html#ansible-pull*）中获取更多关于它的信息。

Ansible 管理小规模集群

没错，Ansible 可以轻松向上扩展至管理成百上千的节点。但是，它真正吸引我的地方是它向下收缩规模的能力。使用 Ansible 管理单一节点非常简单，你只需要编写一个 Ansible 脚本文件。Ansible 遵循了 Alan Kay 的格言“简单的事情应该保持简单，复杂的事情应该做到可能”。

内置模块

你可以使用 Ansible 在你管理的远程服务器上执行任意 Shell 命令，但是 Ansible 真正强大的地方在于内置的一系列模块。通过模块，你可以执行像安装软件包、重启服务或者复制配置文件这样的任务。

稍后我们就会看到，Ansible 模块是*声明式*的，可以使用它们来描述你希望服务器所处于的状态。你可以像下面这样调用 user 模块来确保所有“web”组的服务器上都有“deploy”这个用户：

```
user: name=deploy group=web
```

7 另外，模块是*幂等性*的。如果用户“deploy”不存在，Ansible 就创建它。如果存在，Ansible 不会做任何事。幂等性是个非常赞的特性，因为它意味着向同一台服务器多次执行同一个 Ansible playbook 是安全的。相对于一般运维团队自己编写的 shell 脚本来说，这是一个非常大的优势。运维团队自己编写的脚本在第二次执行的时候很可能会带来不一样的（并且很可能是意外的）影响。

关于收敛性（Convergence）

配置管理领域的书籍往往会提到收敛性的概念。配置管理中的收敛性与 Mark Burgess 以及他编写的配置管理系统 *CFEngine* 最紧密相关。

如果一个配置管理系统是收敛性的，那么这个系统也许需要多次运行才能将服务器置于期望的状态。而在这个过程中的每一次运行，都会使服务器更接近于那个状态。

收敛性的想法并没有被真正应用到 Ansible 中，Ansible 并没有需要多次运行来配置服务器的设计。与之相对，Ansible 的模块实现的行为是幂等性（idempotence）的，只需要运行一次 playbook 就可以将每台服务器都置为期望的状态。

如果你对 Ansible 作者对于收敛性理念的看法感兴趣，可以看看 Michael DeHaan 发布在 Ansible 项目新闻组（*https://groups.google.com/forum/#!msg/ansible-project/WpRblldA2PQ/lYDpFjBXDlsJ*）的文章，题目为：*Idempotence, convergence, and other silly fancy words we use to often*。

非常轻量的抽象层

某些配置管理工具提供一个抽象层，这样你就可以对运行不同操作系统的服务器使用相同的配置管理脚本来管理。例如，配置管理工具提供一个“package”抽象去替代 yum 或者 apt 等包管理工具，这样就无须处理包管理工具之间的差异了。

Ansible 的处理方式不太一样。在基于 apt 的系统上，你还是得使用“apt”模块安装软件包，在基于 yum 的系统上使用“yum”模块安装软件包。

虽然听起来这种设计是个缺点，但从实践角度来看，我认为它反倒使得 Ansible 更易用。Ansible 不需要我们去学习一套新的用于屏蔽不同操作系统差异的抽象。这使得 Ansible 更易上手，在你开始使用之前几乎不需要什么其他知识。

8

如果你愿意，可以在编写自己的 Ansible playbook 时，实现针对运行不同操作系统的远程服务器执行不同的操作。但是实际工作中我会尽量避免这么做。相反，我会更专注于编写用于某个特定操作系统（比如 Ubuntu）的 playbook。

在 Ansible 社区，复用的基本单元是模块。由于模块的功能范围非常小，并且可以只针对特定的操作系统，所以非常易于实现定义明确且易于分享的模块。Ansible 项目对于接受社区贡献的模块这点上非常开放。我也贡献过几个模块，因此很清楚这一点。

Ansible playbook 的设计并不是为了在不同的系统环境之间复用。在第 8 章我们将会讨论 *role*。role 是一种整合 playbook 的方法，所以 role 是可复用的。此外还有非常便利的 Ansible Galaxy，它是一个在线的 role 仓库。

在实际中，每家公司组建服务的方式都略有不同，针对自己公司的情况来编写 playbook 会比尝试复用别人的 playbook 更合适。但是我认为查阅其他人分享的 playbook 仍然具有重要价值，可以帮助理解工作原理。

Ansible 软件与 Ansible 公司是什么关系

Ansible 不仅是软件的名字，还是运作这个开源软件的公司的名字。Ansible 软件的创始人 Michael DeHaan 同时也是 Ansible 公司的前 CTO。为了防止混淆，我们将把软件称作 Ansible，而公司则使用“Ansible 公司”来称呼。

Ansible 公司主要经营围绕 Ansible 的培训和咨询服务。除此之外，还销售一个名为 *Ansible Tower* 的基于 Web 的管理工具。

Ansible 太过于简单了吗

在我撰写这本书的时候，我的编辑曾经向我提起有些使用“某某配置管理工具”的人说 Ansible 就是基于 SSH 的 for 循环脚本。如果你也在考虑是否要从其他配置管理工具切换到 Ansible，你可能也会担心 Ansible 的功能是否足够强大以满足你的需求。

你马上就会看到，Ansible 提供了大量 Shell 脚本难以实现的功能。除了我之前提到的提供幂等性的模块之外，Ansible 对于模板提供了完美的支持，而且在不同的范围内定义变量。任何将使用 Ansible 与使用 Shell 脚本画等号的人，大概都没有不得不维护一个由 Shell 编写的重要程序的经历。如果让我来选择用于完成配置管理任务的工具，我永远都会选择 Ansible 而不是 Shell 脚本。

9

担心 SSH 的扩展性？我们会在第 9 章将详细讨论这个问题。Ansible 使用 SSH multiplexing 来优化性能。并且，业界中已经有人使用 Ansible 来管理成千上万的节点[注3]。

我对于其他配置管理工具并不是十分熟悉，所以无法在此比较它们的细节区别。如果你想了解针对配置管理工具的细致比较，我推荐你看看 Matt Jaynes 写的 *Taste Test: Puppet, Chef, Salt, Ansible*。十分巧合，Matt 也是 Ansbile 党。

注 3： 可以参阅 Rackspace 的 Jesse Keating 的 *Using Ansible at Scale to Manage a Public Cloud*（*http://www.slideshare.net/JesseKeating/ansiblefest-rax*）。

我需要具备哪些基础知识

想要更高效地使用 Ansible 的话，你需要对基本的 Linux 系统管理非常熟悉。Ansible 可以帮你非常轻松地实现任务自动化，但它不是能将你自己都不知道如何做的工作做完的“自动化魔法”。

具体到本书，我会假定读者们至少熟悉某一个 Linux 发行版（例如 Ubuntu、RHEL、CentOS、SUSE），并且懂得如何完成如下工作：

- 使用 SSH 连接到远程服务器。
- 会处理 Bash 命令行的输入输出（管道和重定向）。
- 安装软件包。
- 使用 sudo 命令。
- 检查和设置文件权限。
- 启动和停止系统服务。
- 设置环境变量。
- 编写脚本（语言不限）。

如果你熟悉上面的所有概念，那么你就已经准备好学习和使用 Ansible 了。

我不会假定你对某一特定编程语言有所了解。例如，你根本不需要会写 Python 就可以使用 Ansible。但如果你希望能够编写自己的模块就另当别论了。

Ansible 使用 YAML 文件格式和 Jinja2 模板语言，所以要使用 Ansible 你需要学习一些 YAML 和 Jinja2 的知识，它们都很容易上手。

哪些内容不会涉及

本书并不是 Ansible 百科全书。本书编写的目的是为了帮你快速上手 Ansible，并演示一些执行任务的典型范例，这些范例都是无法直接在官方文档中查阅到的。

本书不会详细介绍所有 Ansible 官方模块。Ansible 有 200 多个官方模块，并且 Ansible 的模块的官方参考文档已经很棒了。

Ansible 使用 Jinja2 作为模板引擎，我仅会在本书介绍模板引擎的基本功能。这主要是因为在使用 Ansible 的过程中，我发现通常只需要用到 Jinja2 的基本功能。如果在你的模板中需要使用 Jinja2 的更多高级功能，建议查阅 Jinja2 的官方文档（*http://jinja.pocoo.org/docs/dev/*）。

Ansible 的有些特性主要是为了更好地在旧版本 Linux 上运行，本书不会详细介绍这些特性。这些特性包括 *paramiko SSH* 客户端和 *accelerated mode*。对于这些特性相关的问题，我会给出官网文档的链接并简单略过。

在 1.7 版本中，Ansible 加入了支持管理 Windows 服务器的特性。我不会在本书中介绍关于 Windows 的内容，因为我并没有使用 Ansible 管理 Windows 服务器的经验。并且我始终认为这是一个细分场景，使用 Ansible 管理 Windows 主机领域可能适合单独写一本书。

我不会在本书讨论 Ansible Tower，它是由 Ansible 公司开发的，用于管理 Ansible 的商业 Web 工具。本书将聚焦在 Ansible 本身。而 Ansible 及其所附带的所有模块都是完全开源的。

最后，还有几个 Ansible 的特性被我忽略，以保持书的厚度可控。这些特性包括拉取模式、日志、使用非 SSH 协议连接到主机，以及交互式输入信息或密码。建议你查阅官方文档来了解这些特性的更多信息。

安装 Ansible

如果你在 Linux 上运行 Ansible，目前所有主流的 Linux 发行版都有 Ansible 的软件包，所以你可以使用原生包管理工具安装它。但是这种方法安装的 Ansible 可能是一个旧版本。如果你在 Mac OS X 上运行 Ansible，我强烈建议你使用非常赞的 Homebrew 包管理工具来安装 Ansible。

如果所有其他的方法都无法使用，你可以使用 Python 包管理工具 *pip* 来安装 Ansible。你可以用如下方法使用 root 来安装它：

```
$ sudo pip install ansible
```

11 如果你不想使用 root 来安装 Ansible，则可以将 Ansible 安全地安装到一个 Python *virtualenv* 中。如果你对于 Python virtualenv 不熟悉，可以使用一个比较新的工具 *pipsi*，它可以帮你自动将 Ansible 安装到 Python virtualenv 中：

```
$ wget https://raw.githubusercontent.com/mitsuhiko/pipsi/master/get-pipsi.py
$ python get-pipsi.py
$ pipsi install ansible
```

如果你选择使用 pipsi 来安装，就需要将 *~/.local/bin* 添加到你的 PATH 环境变量中。某些 Ansible 的插件或者模块可能会依赖额外的 Python 库。如果你已经安装了 pipsi，并且想要安装 *docker-py*（Ansible 的 Docker 模块需要此 Python 库）和 *boto*（Ansible 的 EC2

模块需要此 Python 库），则需要进行如下操作：

```
$ cd ~/.local/venvs/ansible
$ source bin/activate
$ pip install docker-py boto
```

如果你比较喜欢冒险，希望使用最前沿版本的 Ansible，你甚至可以直接从 GitHub 上获取开发版分支：

```
$ git clone https://github.com/ansible/ansible.git --recursive
```

如果你选择使用开发版分支的 Ansible，则每次都需要使用如下命令更新环境变量。这其中也包括 PATH 环境变量，以便 shell 知道 *ansible* 和 *ansible-playbook* 程序的具体位置。

```
$ cd ./ansible
$ source ./hacking/env-setup
```

想了解关于安装的更多细节，可以查看如下资源。

- Ansible 官方安装文档：*http://docs.ansible.com/intro_installations.html*。
- pip：*http://pip.readthedocs.org/*。
- virtualenv：*http://docs.python-guide.org/en/latest/dev/virtualenvs/*。
- pipsi：*https://github.com/mitsuhiko/pipsi* 。

建立一台用于测试的服务器

如果想要跟着本书中的范例来做试验的话，你需要拥有一台 Linux 服务器并具有 SSH 登录和 root 权限。幸运的是，目前通过公有云服务登录一台低成本的 Linux 虚拟机非常简单便利，例如 Amazon EC2、Google Compute Engine、Microsoft Azure[注4]、Digital Ocean、Rackspace、SoftLayer、HP Public Cloud、Linode 等。

12

使用 Vagrant 来创建测试服务器

如果你不想花钱购买公有云服务，我建议你在自己的服务器上安装 Vagrant。Vagrant 是一个优秀的开源虚拟机管理工具。你可以使用 Vagrant 在你的笔记本电脑上启动一台 Linux 虚拟机，我们可以使用它作为测试服务器。

Vagrant 内置支持使用 Ansible 部署虚拟机，但是我们将在第 11 章再详细讨论这个话题。现在，我们仅仅是将 Vagrant 虚拟机当作一台普通的 Linux 服务器来管理。

注 4：　是的，你没有看错。Azure 提供 Linux 服务器。

要使用 Vagrant，需要保证你的机器上已经安装了 VirtualBox。VirtualBox 可以在 *http://www.virtualbox.org* 下载，而 Vagrant 则可以在 *http://www.vagrantup.com* 下载。

我建议你专门创建一个目录用于存储 Ansible playbook 和相关文件。在后面的范例中，我将这个目录命名为 *playbooks*。

运行下列命令将创建一个 64 位 Ubuntu 14.04（Trusty Tahr）[注5] 虚拟机镜像[注6] 对应的 Vagrant 配置文件（Vagrantfile），并启动该虚拟机。

```
$ mkdir playbooks
$ cd playbook
$ vagrant init ubuntu/trusty64
$ vagrant up
```

当你第一次运行 `vagrant up` 的时候，它将会自动下载虚拟机镜像文件。因此第一次运行 `vagrant up` 将会需要多一些的时间，具体时长取决于网络连接的速度。

如果一切正常，输出将会如下所示：

```
A `Vagrantfile` has been placed in this directory. You are now
    ready to `vagrant up` your first virtual environment! Please read
    the comments in the Vagrantfile as well as documentation on
    `vagrantup.com` for more information on using Vagrant.

    Bringing machine'default'up with'virtualbox'provider...
    ==> default: Box'ubuntu/trusty64'could not be found. Attempting to find
    and install...
        default: Box Provider: virtualbox
        default: Box Version: >= 0
    ==> default: Loading metadata for box'ubuntu/trusty64'
        default: URL: https://vagrantcloud.com/ubuntu/trusty64
    ==> default: Adding box'ubuntu/trusty64'(v14.04) for provider: virtualbox
        default: Downloading: https://vagrantcloud.com/ubuntu/trusty64/
        version/1/provider/virtualbox.box
    ==> default: Successfully added box'ubuntu/trusty64'(v14.04) for
        'virtualbox'!
    ==> default: Importing base box'ubuntu/trusty64'...
    ==> default: Matching MAC address for NAT networking...
```

13

注5： 每个 Ubuntu 版本都会拥有一个由“形容词＋动物”组成的代号。Trusty Tahr 是 Ubuntu 14.04 的代号。——译者注

注6： Vagrant 使用术语 *machine* 代表虚拟机，术语 *box* 代表虚拟机镜像。

```
==> default: Checking if box'ubuntu/trusty64'is up to date...
==> default: Setting the name of the VM: vm_default_1409457679518_47647
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection timeout. Retrying...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
==> default: Mounting shared folders...
    default: /vagrant => /Users/lorinhochstein/playbooks
```

你可以运行如下命令 SSH 到新出炉的 Ubuntu 10.04 虚拟机中：

```
$ vagrant ssh
```

如果一切顺利，你应当看到如下所示的登录信息：

```
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-35-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Sun Aug 31 04:07:21 UTC 2014

System load:  0.0                Processes:           73
Usage of /:   2.7% of 39.34GB    Users logged in:     0
Memory usage: 25%                IP address for eth0: 10.0.2.15
Swap usage:   0%

      Graph this data and manage this system at:
        https://landscape.canonical.com/

      Get cloud support with Ubuntu Advantage Cloud Guest:
        http://www.ubuntu.com/business/services/cloud

    0 packages can be updated.
    0 updates are security updates.
Last login: Sun Aug 31 04:07:21 2014 from 10.0.2.2
```

14

输入 **exit** 退出 SSH 会话。

这种方法让我们直接与 Shell 交互，但是 Ansible 使用标准 ssh 客户端连接到虚拟机，而不是使用 `vagrant ssh` 命令。

如下操作可以告诉 Vagrant 输出 SSH 连接的详细信息：

```
$ vagrant ssh-config
```

在我的机器上，输出信息如下：

```
Host default
  HostName 127.0.0.1
  User vagrant
  Port 2222
  UserKnownHostsFile /dev/null
  StrictHostKeyChecking no
  PasswordAuthentication no
  IdentityFile /Users/lorinhochstein/dev/ansiblebook/ch01/playbooks/.vagrant/
  machines/default/virtualbox/private_key
  IdentitiesOnly yes
  LogLevel FATAL
```

其中最重要的部分有：

```
  HostName 127.0.0.1
  User vagrant
  Port 2222
  IdentityFile /Users/lorinhochstein/dev/ansiblebook/ch01/playbooks/.vagrant/
  machines/default/virtualbox/private_key
```

Vagrant 1.7 版本改变了它处理 SSH 私钥的行为。在 Vagrant 1.7 版本中，Vagrant 为每个机器都创建了一个私钥。之前的版本使用相同的私钥，该私钥存放在 *~/.vagrant.d/insecure_private_key* 中。本书中的范例都使用 Vagrant 1.7。

在你的机器上，除了 IdentifyFile 的路径之外，其他每个部分都应该与范例中一样。

这些信息可以用来确认你可以从命令行发起 SSH 会话。在我的机器上，SSH 命令是这样的：

```
$ ssh vagrant@127.0.0.1 -p 2222 -i /Users/lorinhochstein/dev/ansiblebook/
ch01/
playbooks/.vagrant/machines/default/virtualbox/private_key
```

你应该能看到 Ubuntu 的登录界面。输入 **`exit`** 退出 SSH 会话。

将测试服务器的信息配置在 Ansible 中

15

Ansible 只能管理那些它明确了解的服务器。你只需要在 inventory 文件中指定服务器的信息就可以将这些信息提供给 Ansible。

每台服务器都需要一个用来让 Ansible 识别的名字。你可以使用服务器的主机名，或者也可以给服务器起一个别名，并传递一些附加参数来告诉 Ansible 如何连接到服务器的。我们将为 Vagrant 服务器起一个别名：`testserver`。

在 *playbooks* 目录下创建一个名叫 *hosts* 的文件。这个文件将作为 inventory 文件。如果你正在使用 vagrant 虚拟机作为测试服务器，你的 *hosts* 文件应该与例 1-1 所示内容很相像。我把文件内容拆分到多行是因为页面的限制，在你的文件中这些内容应该是一行。

例1-1　playbooks/hosts

```
testserver ansible_ssh_host=127.0.0.1 ansible_ssh_port=2222 \
 ansible_ssh_user=vagrant \
 ansible_ssh_private_key_file=.vagrant/machines/default/virtualbox/private_key
```

这里我们看到一个使用 Vagrant 的缺点。我们不得不明确地传入额外参数来告诉 Ansible 如何连接。在绝大部分情况下，我们不需要这些额外信息。

在本章稍后部分，我们将会看到如何通过使用 *ansible.cfg* 文件来避免冗长的 inventory 文件。在后面的章节，我们还将看到如何利用 Ansible 变量达到相近的效果。

如果你有一台运行在 Amazon EC2 的 Ubuntu 机器，主机名类似 `ec2-203-0-113-120.compute-1.amazonaws.com` 这样。那么你的 inventory 文件将类似如下所示（所有信息应该在一行）：

```
testserver ansible_ssh_host=ec2-203-0-113-120.compute-1.amazonaws.com \
ansible_ssh_user=ubuntu ansible_ssh_private_key_file=/path/to/keyfile.pem
```

Ansible 支持 ssh-agent 程序，所以你不需要在你的 inventory 文件中明确说明 SSH 文件。如果你以前没有使用过 ssh-agent，请查看在 279 页的“SSH agent”获取更详细信息。

我们将使用 `ansible` 命令行工具来验证我们是否可以使用 Ansible 连接到服务器。你不会经常用到 `ansible` 命令，它通常用于点对点的、一次性的事情上。

让我们告诉 Ansible 连接到名为 `testserver` 的服务器（在 inventory 的 *hosts* 文件中描述，

16

并且引入 ping 模块：

```
$ ansible testserver -i hosts -m ping
```

如果你的本地 SSH 客户端打开了 host key 验证功能，那么在 Ansible 第一次尝试连到主机时，你可能会看到如下信息：

```
The authenticity of host'[127.0.0.1]:2222 ([127.0.0.1]:2222)'\ can't be
established. RSA key fingerprint is e8:0d:7d:ef:57:07:81:98:40:31:19:53:a8
:d0:76:21. Are you sure you want to continue connecting (yes/no)?
```

直接输入 **yes** 就好。

如果执行成功，将会得到如下所示的输出：

```
testserver | success >> {
    "changed": false,
    "ping":"pong"
}
```

如果 Ansible 的执行没有成功，可以添加 `-vvvv` 参数来查看这个错误的更多细节。

```
$ ansible testserver -i hosts -m ping -vvvv
```

我们可以看到模块执行成功了。输出中 `"changed":false` 部分告诉我们模块的执行没有改变服务器的状态。输出中 `"ping":"pong"` 是由 *ping* 模块定义的输出。

ping 模块除了检查 Ansible 是否可以打开到服务器的 SSH 会话之外，并不做任何其他的事情。它就是用来检测是否能连接到服务器的实用工具。

使用 ansible.cfg 文件来简化配置

在前面的范例中，我们不得不在 inventory 文件中输入很多内容来告诉 Ansible 关于测试服务器的信息。幸好 Ansible 有许多方法来让你定义各种变量。这样，我们就不需要把那些信息都堆在一个地方了。

我们马上就会使用一个这样的机制，*ansible.cfg* 文件。*ansible.cfg* 文件可以设定一些默认值，这样我们就不需要对同样的内容输入很多遍。

我应该把 ansible.cfg 文件放到哪里呢

Ansible 按照如下位置和顺序来查找 *ansible.cfg* 文件：

1. `ANSIBLE_CONFIG` 环境变量所指定的文件。
2. *./ansible.cfg*（当前目录下的 *ansible.cfg*）。
3. *~/.ansible.cfg*（主目录下的 *.ansible.cfg*）。
4. */etc/ansible/ansible.cfg*。

我通常把 *ansible.cfg* 与我的 playbooks 一起放到当前目录。这样，我就可以把它和 playbooks 放到同一个版本控制仓库中。

例 1-2 展示的 *ansible.cfg* 文件指定了 inventory 文件的位置（*hostfile*）、SSH 使用的用户名（*remote_user*）和 SSH 私钥（*private_key_file*）。这个范例中假定你在使用 Vagrant。如果你在使用自己的服务器，你需要将 *remote_user* 和 *private_key_file* 设为相应的值。

例1-2　ansible.cfg

```
[defaults]
hostfile = hosts
remote_user = vagrant
private_key_file = .vagrant/machines/default/virtualbox/private_key
host_key_checking = False
```

我们的范例配置还关闭了 SSH 的 host key 检查。这样当处理 Vagrant 机器时就会很方便，否则，每次我们销毁并重新创建一个 Vagrant 虚拟机之后，都需要编辑我们的 *~/.ssh/known_hosts* 文件。但是，关闭主机秘钥检查在通过网络连接其他主机时会成为安全风险。如果你对 host key 并不熟悉，请查看附录 A 中关于 host key 的更多细节。

Ansible 与版本控制

Ansible 使用 */etc/ansible/hosts* 作为 inventory 文件的默认位置。不过，我从来不使用它，因为我希望我的 inventory 文件和 playbook 一起进行版本控制。

虽然本书没有覆盖到版本控制的内容，我仍然强烈建议你使用类似 Git 这样的版本控制系统来保管所有的 playbook。如果你是一位开发者，那么你应该对配置管理系统已经相当熟悉了。如果你是系统管理员并且还没有使用过版本控制系统，这恰好是个完美的上手时机。

有了我们设定的默认值，我们再也不需要在 *hosts* 文件中指定 SSH 用户和私钥文件了。它将简化为：

```
testserver ansible_ssh_host=127.0.0.1 ansible_ssh_port=2222
```

我们还可以在执行 ansible 命令时去掉 `-i hostname` 参数：

```
$ ansible testserver -m ping
```

我喜欢使用 *ansible* 命令行工具随心所欲地在远程机器上执行命令，就好像并行执行 SSH 脚本一样。你也可以使用 *command* 模块来随心所欲地执行命令。当使用这个模块的时候，你需要使用 `-a` 参数将需要执行的命令传入模块。

例如，想要查看服务器上的运行时间可以这么实现：

```
$ ansible testserver -m command -a uptime
```

输出应该类似如下内容：

```
testserver | success | rc=0 >>
17:14:07 up  1:16,  1 user,  load average: 0.16, 0.05, 0.04
```

command 模块很常用，因而 ansible 命令将它设为了默认模块。所以我们也可以使用更简单的方式执行：

```
$ ansible testserver -a uptime
```

如果我们的命令包含空格，我们需要使用引号将命令括起来，这样 shell 才会将整个字符串作为单一参数传给 Ansible。例如，要查看 */var/log/dmesg* 日志文件的最后几行：

```
$ ansible testserver -a"tail /var/log/dmesg"
```

我的 Vagrant 虚拟机上的输出类似下面这样：

```
testserver | success | rc=0 >>
[    5.170544] type=1400 audit(1409500641.335:9): apparmor="STATUS"
operation="proile_replace" profile="unconfined" name="/usr/lib/NetworkManager/
nm-dhcp-client.act on" pid=888 comm="apparmor_parser"
[    5.170547] type=1400 audit(1409500641.335:10): apparmor="STATUS"
operation="pro file_replace" profile="unconfined" name="/usr/lib/connman/
scripts/dhclient-script"pid=888 comm="apparmor_parser"
[     5.222366] vboxvideo: Unknown symbol drm_open (err 0)
[     5.222370] vboxvideo: Unknown symbol drm_poll (err 0)
[     5.222372] vboxvideo: Unknown symbol drm_pci_init (err 0)
[     5.222375] vboxvideo: Unknown symbol drm_ioctl (err 0)
[     5.222376] vboxvideo: Unknown symbol drm_vblank_init (err 0)
[     5.222378] vboxvideo: Unknown symbol drm_mmap (err 0)
```

```
[     5.222380] vboxvideo: Unknown symbol drm_pci_exit (err 0)
[     5.222381] vboxvideo: Unknown symbol drm_release (err 0)
```

如果我们需要使用root来执行，需要传入参数 `-s` 来告诉Ansible要*sudo*为root执行。例如，访问 */var/log/syslog* 需要使用 root 权限：

```
ansible testserver -s -a "tail /var/log/syslog"
```

输出如下：

```
testserver | success | rc=0 >>
Aug 31 15:57:49 vagrant-ubuntu-trusty-64 ntpdate[1465]:/
adjust time server 91.189
94.4 offset -0.003191 sec
Aug 31 16:17:01 vagrant-ubuntu-trusty-64 CRON[1480]: (root) CMD (   cd / &&
run-p
rts --report /etc/cron.hourly)
Aug 31 17:04:18 vagrant-ubuntu-trusty-64 ansible-ping: Invoked with data=None
Aug 31 17:12:33 vagrant-ubuntu-trusty-64 ansible-ping: Invoked with data=None
Aug 31 17:14:07 vagrant-ubuntu-trusty-64 ansible-command: Invoked with
executable
None shell=False args=uptime removes=None creates=None chdir=None
Aug 31 17:16:01 vagrant-ubuntu-trusty-64 ansible-command: Invoked with
executable
None shell=False args=tail /var/log/messages removes=None creates=None
chdir=None
Aug 31 17:17:01 vagrant-ubuntu-trusty-64 CRON[2091]: (root) CMD (   cd / &&
run-pa
rts --report /etc/cron.hourly)
Aug 31 17:17:09 vagrant-ubuntu-trusty-64 ansible-command: Invoked with
executable=
N one shell=False args=tail /var/log/dmesg removes=None creates=None
chdir=None
Aug 31 17:19:01 vagrant-ubuntu-trusty-64 ansible-command: Invoked with
executable=
None shell=False args=tail /var/log/messages removes=None creates=None
chdir=None
Aug 31 17:22:32 vagrant-ubuntu-trusty-64 ansible-command: Invoked with
executable=
one shell=False args=tail /var/log/syslog removes=None creates=None
chdir=None
```

我们可以从输出中看到，Ansible 在运行时会写入 syslog。

在使用 *ansible* 命令行工具的时候，不仅可以使用 *ping* 和 *command* 模块，还可以使用任何你喜欢的其他模块。例如，你可以像这样在 Ubuntu 上安装 nginx：

```
$ ansible testserver -s -m apt -a name=nginx
```

如果安装 nginx 失败，那么你可能需要更新一下软件包列表。为了告诉 Ansible 在安装软件包之前执行等同于 `apt-get update` 的操作，需要将参数从 `name=nginx` 变为 `"name=nginx update_cache=yes"`。

你可以使用如下操作重启 nginx：

```
$ ansible testserver -s -m service -a name=nginx
  state=restarted
```

20 因为只有 root 才可以安装 nginx 软件包和重启服务，所以我们需要使用 `-s` 参数来使用 sudo。

继续前进

简要回顾一下：在本章，我们简要介绍了 Ansible 的基本概念，包括它如何与远程服务器通信，以及它和其他配置管理工具的不同之处。我们还演示了如何使用 *ansible* 命令行工具对单台主机执行简单任务。

当然了，使用 *ansible* 对单台主机运行命令并不怎么有趣。在下一章，我们将讲解 playbook，这才是重头戏。

第 2 章 21

playbook：一切的开端

使用 Ansible 时，绝大部分时间将花费在编写 *playbook* 上。playbook 是一个 Ansible 术语，它指的是用于配置管理的脚本。让我们看一个实例：安装 Nginx Web 服务器并将其配置为可用于安全通信的状态。

如果你完全遵从本章的范例来操作，那么最后你会完成的文件列表如下：

- *playbooks/ansible.cfg*
- *playbooks/hosts*
- *playbooks/Vagrantfile*
- *playbooks/web-notls.yml*
- *playbooks/web-tls.yml*
- *playbooks/files/nginx.key*
- *playbooks/files/nginx.crt*
- *playbooks/files/nginx.conf*
- *playbooks/templates/index.html.j2*
- *playbooks/templates/nginx.conf.j2*

一些准备工作

在我们对 Vagrant 虚拟机运行这个 playbook 之前，需要先暴露 80 和 443 端口以便访问它们。如图 2-1 所示，我们将通过配置 Vagrant 实现对本地机器的 8080 端口和 8443 端口的请求转发到 Vagrant 虚拟机的 80 端口和 443 端口。这样我们就可以通过 *http://localhost:8080* 和 *http://localhost:8443* 访问到运行在 Vagrant 里的 Web 服务器。 22

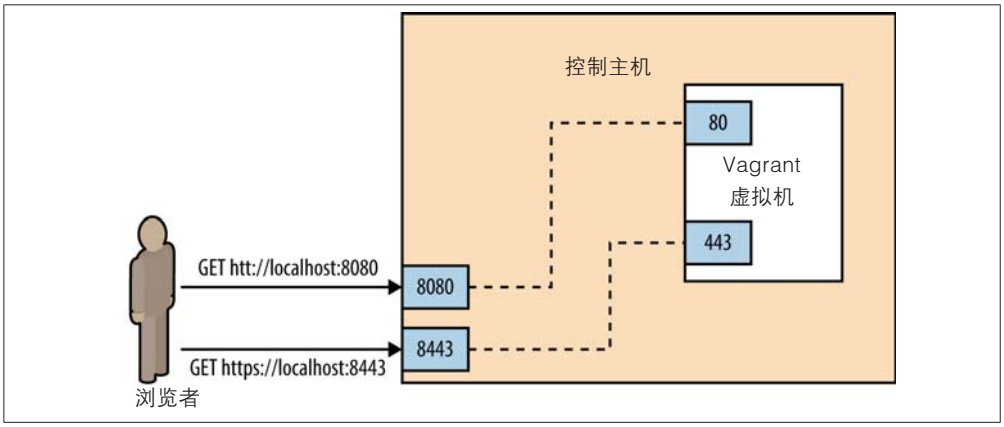


图2-1　暴露Vagrant虚拟机上的端口

如下所示修改你的 *Vagrantfile* ：

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.network "forwarded_port", guest: 443, host: 8443
end
```

这会将本机的 8080 端口映射到 Vagrant 虚拟机的 80 端口，并将本机的 8443 端口映射到 Vagrant 虚拟机的 443 端口。完成变更后，需要执行如下操作来告诉 Vagrant 让新配置生效 ：

```
$ vagrant reload
```

你会看到输出包括如下内容 ：

```
==> default: Forwarding ports...
    default: 80 => 8080 (adapter 1)
    default: 443 => 8443 (adapter 1)
    default: 22 => 2222 (adapter 1)
```

一个简单的 playbook

在我们的第一个 playbook 范例中，我们将配置一台主机来运行 Nginx Web 服务器。在这个范例中，我们不会把 Web 服务器配置为支持 TLS 加密的，这将使得建立 Web 服务器更简单些。但是真正的网站应该启用 TLS 加密，我们将在本章稍后的部分介绍如何实现它。

23 首先，我们来看一下当运行例 2-1 中的代码后会发生什么，然后我们会详细分析这一段代码。

例2-1　web-notls.yml

```
- name: Configure webserver with nginx
  hosts: webservers
  sudo: True
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes

    - name: copy nginx config file
      copy: src=files/nginx.conf dest=/etc/nginx/sites-available/default

      - name: enable configuration
      file: >
        dest=/etc/nginx/sites-enabled/default
        src=/etc/nginx/sites-available/default
        state=link

    - name: copy index.html
      template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html
      mode=0644

    - name: restart nginx
      service: name=nginx state=restarted
```

为什么在有的地方使用“True”，而其他地方又使用“yes”

眼尖的读者可能已经注意到例 2-1 中，在 playbook 中的某处使用了 True（用来启用 sudo），又在 playbook 中的另一处使用了 yes（用来更新 apt 缓存）。

关于如何在 playbook 中表示“是”和“否”，Ansible 非常灵活。严格地说，模块参数（就像 update_cache=yes）对于值的处理和 playbook 中其他地方（就像 sudo:True）对于值的处理有所不同。这是因为其他地方由 YAML 解析器来处理值，所以只用 YAML 的真实性值的约定：

YAML“真”值

true，True，TRUE，yes，Yes，YES，on，On，ON，y，Y

YAML“否”值

false，False，FALSE，no，No，NO，off，Off，OFF，n，N

模块参数是作为字符串传递的，因此使用 Ansible 内置约定：

模块参数为真

yes，on，1，true

模块参数为假

no，off，0，false

我倾向于遵从 Ansible 官方文档的示例。向模块传递参数时候使用 yes 和 no（这是为了保持和模块官方文档一致），而 playbook 中其他地方使用 True 和 False。

指定一个 nginx 配置文件

在这个 playbook 可以运行之前，它还需要两个额外的文件。首先，我们需要定义一个 nginx 的配置文件。

如果你只需要提供静态服务内容的话，nginx 随附的那个开箱即用的文件就挺好的。但是绝大多数情况下都还是需要自定义一部分配置，所以我们还是需要用自己的配置文件覆盖默认的文件，并将它作为 playbook 的一部分。就像我们后面部分将会看到的，我们将需要修改 nginx 配置文件以添加对 TLS 的支持。例 2-2 是一个 nginx 的基本配置文件，将它保存在 *playbooks/files/nginx.conf* 中[注1，注2]。

例2-2　files/nginx.conf

```
server {
        listen 80 default_server;
        listen [::]:80 default_server ipv6only=on;

        root /usr/share/nginx/html;
        index index.html index.htm;

        server_name localhost;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

按照惯例，Ansible 会将一般文件放在名为 files 的子目录中，将 Jinja2 模板文件放在名为 *templates* 的子目录中。我将在整本书中都遵循这个惯例。

注 1：　需要注意的是，虽然我们叫这个文件 *nginx.conf*，但实际上它替代的是 nginx 子配置文件。

注 2：　按 nginx 默认的配置文件分割习惯，这个文件是配置虚拟主机的。在 nginx 中虚拟主机的术语是 Server Block)——译者注

创建一个定制的首页

让我们来添加一个定制的首页。我们将使用 Ansible 的模板功能，使 Ansible 从模板中生成这个配置文件。模板如例 2-3 所示，将它放到 *playbooks/templates/index.html.j2* 中。

例2-3　playbooks/template/index.html.j2

```
<html>
  <head>
    <title>Welcome to ansible</title>
  </head>
  <body>
  <h1>nginx, configured by Ansible</h1>
  <p>If you can see this, Ansible successfully installed nginx.</p>
  <p>{{ ansible_managed }}</p>
  </body>
</html>
```

这个模板引用了一个特别的 Ansible 变量，叫作 *ansible_managed*。当 Ansible 渲染这个模板的时候，它将会把这个变量替换为和这个模板文件生成时间相关的信息。图 2-2 展示了一张在浏览器中查看生成的 HTML 页面时的截图。

图2-2　渲染后的HTML

创建一个 webservers 群组

让我们在 inventory 文件中创建一个 webservers 群组，以便在 playbook 中引用这个群组。目前，这个群组只包含我们的测试服务器。

inventory 文件使用 *.ini* 文件格式。我们将在本书后面的章节深入研究这种格式。编辑你的 *playbooks/hosts* 文件，在 testserver 行上面添加一行 `[webservers]`，如例 2-4 所示。这表示这台 `testserver` 属于 webservers 群组。

例2-4 playbooks/hosts

```
[webservers]

testserver ansible_ssh_host=127.0.0.1 ansible_ssh_port=2222
```

现在你应该可以使用 ansible 命令行工具来 ping webservers 群组了。

```
$ ansible webservers -m ping
```

输出应该看起来像这样：

```
testserver | success >> {
"changed": false,

    "ping": "pong"
}
```

运行这个 playbook

执行 playbook 需要使用 ansible-playbook 命令。按照如下方法运行 playbook：

```
$ ansible-playbook web-notls.yml
```

输出应该如例 2-5 所示：

例2-5 ansible-playbook的输出

```
PLAY [Configure webserver with nginx] *********************************

GATHERING FACTS ***************************************************************
ok: [testserver]

TASK: [install nginx] *********************************************************
changed: [testserver]

TASK: [copy nginx config file] ************************************************
changed: [testserver]

TASK: [enable configuration] **************************************************
ok: [testserver]

TASK: [copy index.html] *******************************************************
changed: [testserver]

TASK: [restart nginx] *********************************************************
changed: [testserver]

PLAY RECAP ********************************************************************
testserver                 : ok=6    changed=4    unreachable=0    failed=0
```

27

Cowsay

如果你在本地机器安装了 *cowsay* 程序，Ansible 的输出将会变成如下所示：

```
 _______________________________________
< PLAY [Configure webserver with nginx] >
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

如果你不想看到这头奶牛，你可以通过设置 `ANSIBLE_NOCOWS` 环境变量来关闭 cowsay。

```
$ export ANSIBLE_NOCOWS=1
```

你还可以通过在 *ansible.cfg* 中添加如下一行来关闭 cowsay。

```
[defaults]
nocows = 1
```

如果你没碰到任何报错[注3]，你应能使用浏览器访问 *http://localhost:8080* 并看到自定义的 HTML 页面，输出结果如图 2-2 所示。

如果你将 playbook 文件权限设置为可执行，并且首行如下所示[注4, 注5]：

```
#!/usr/bin/env ansible-playbook
```

这样你就可以通过直接调用它自己来执行，如下所示：

```
$ ./web-notls.yml
```

28

“Gathering Facts”是做什么用的

你可能已经注意到在 Ansible 初次开始运行的时候有如下输出：

```
GATHERING FACTS ***************************************************
```

注 3：如果遇到了错误，你可能需要跳到第 14 章来帮助你调试。

注 4：这种语法特性叫作 shebang。

注 5：当类 Unix 操作系统中的文本文件第一个行前两个字符为 #! 时叫作 shebang。操作系统的程序载入器会分析 #! 后的内容，将这些内容作为解释器指令，调用该指令，并将这个含有 shebang 的文件及其路径作为该解释器的参数。——译者注

```
ok: [testserver]
```

当 Ansible 开始运行 playbook 的时候，它做的第一件事就是从它连接到的服务器上收集各种信息。这些信息包括：操作系统、主机名、所有网络接口的 IP 地址和 MAC 地址等。

这样，你就可以在之后的 playbook 中使用这些信息了。例如，你可能需要将主机的 IP 地址填充在配置文件中。

如果你不需要使用这些信息，你可以关闭 *fact gathering* 以节省一些时间。在后面的章节中我们会讨论如何使用这些信息以及如何关闭 fact gathering。

playbook 是 YAML 格式的

Ansible 的 playbook 是使用 YAML 语法编写的。YAML 是一种类似于 JSON 的文件格式，不过 YAML 更适合人来读写。在我们开始详细讨论 playbook 之前，让我们先了解一下编写 playbook 所必需的 YAML 语法。

文件的起始

YAML 文件以三个减号开头以标记文档的开始：

```
---
```

不过呢，如果你忘记在你的 playbook 文件开头敲着三个减号，并不会影响 Ansible 的运行。

注释

注释以井号开始一直到本行结束，这与 Shell、Python 及 Ruby 是一样的。

```
# This is a YAML comment
```

字符串

一般来说，YAML 字符串并不必使用引号引起来，当然如果你喜欢用引号引起来也没有问题。即便字符串中含有空格，你也完全不需要使用引号。例如，下面是一个 YAML 中的字符串：

29

```
this is a lovely sentence
```

其等价的 JSON 格式为：

```
"this is a lovely sentence"
```

在 Ansible 中，有几种情况比较例外，需要将字符串引起来。这些情况下，通常会引入 {{braces}} 用法，用于变量替代。我们稍后会接触到这种用法。

布尔型

YAML 具有内置的布尔类型，并且提供了多种多样的释义为“是”或“否”的字符串。我们在 22 页讨论过这个内容。

我个人一直在 Ansible playbook 中使用 True 和 False。

例如，这是 YAML 中的布尔型：

```
True
```

等价的 JSON 格式为：

```
true
```

列表

YAML 中的列表就好像 JSON 和 Ruby 中的数组或者 Python 中的列表。严格来说，在 YAML 中叫作数列。但是在这里，为了和 Ansible 官方文档保持一致，我还是叫它们列表。

列表使用减号“-”作为分割符，就像这样：

```
- My Fair Lady
- Oklahoma
- The Pirates of Penzance
```

等价的 JSON 格式为：

```
[
  "My Fair Lady",
  "Oklahoma",
  "The Pirates of Penzance"
]
```

（再次注意：在 YAML 中，即使字符串中含有空格，我们也不必把它们引起来。）

YAML 还为列表支持一种内联格式，就像这样：

```
[My Fair Lady, Oklahoma, The Pirates of Penzance]
```

30

字典

YAML 中的字典就像 JSON 中的对象、Python 中的字典或者 Ruby 中的哈希。严格来说，在 YAML 里应该叫映射。同样为了与 Ansible 官方文档保持一致，我还是叫它们字典。

字典范例如下：

```
address: 742 Evergreen Terrace
city: Springfield
state: North Takoma
```

等价的 JSON 格式：

```
{
  "address": "742 Evergreen Terrace",
  "city": "Springfield",
  "state":"North Takoma"
}
```

YAML 也为字典支持内联格式，就像下面这样：

```
{address: 742 Evergreen Terrace, city: Springfield, state: North Takoma}
```

折行

编写 playbook 的时候，你会经常碰到需要向模块传递许多参数的场景。为了美观起见，你或许希望将这些很长的段落拆分成多行，同时 Ansible 又能将其视为单行字符串。

你完全可以使用 YAML 语法来实现折行。YAML 中使用大于号（>）来标记折行，YAML 解释器将会把换行符替换为空格。例如：

```
address: >
    Department of Computer Science,
    A.V. Williams Building,
    University of Maryland
city: College Park
state: Maryland
```

等价的 JSON 格式为：

```
{
  "address": "Department of Computer Science, A.V. Williams Building,
              University of Maryland",
  "city": "College Park",
  "state": "Maryland"
}
```

剖析 playbook

让我们从一个 YAML 文件的角度重新审视我们的 playbook。例 2-6 再次列出了 web-notls.yml。

例2-6　web-notls.yml

```
- name: Configure webserver with nginx
  hosts: webservers
  sudo: True
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes

    - name: copy nginx config file
      copy: src=files/nginx.conf dest=/etc/nginx/sites-available/default

    - name: enable configuration
      file: >
        dest=/etc/nginx/sites-enabled/default
        src=/etc/nginx/sites-available/default
        state=link

    - name: copy index.html
      template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html
        mode=0644

    - name: restart nginx
       service: name=nginx state=restarted
```

例 2-7 展示了这个文件的 JSON 版本：

例2-7　web-notls.yml等价的JSON格式

```
[
  {
    "name": "Configure webserver with nginx",
    "hosts": "webservers",
    "sudo": true,
    "tasks": [
      {
        "name": "Install nginx",
        "apt": "name=nginx update_cache=yes"
      },
      {
        "name": "copy nginx config file",
        "template": "src=files/nginx.conf dest=/etc/nginx/
        sites-available/default"
```

```
      },
      {
32      "name": "enable configuration",
        "file": "dest=/etc/nginx/sites-enabled/default src=/etc/nginx/sites-
available/default state=link"
      },
      {
        "name": "copy index.html",
        "template" : "src=templates/index.html.j2 dest=/usr/share/nginx/html/
        index.html mode=0644"
      },
      {
        "name": "restart nginx",
        "service": "name=nginx state=restarted"
      }
    ]
  }
]
```

一个合法的 JSON 文件也一定是一个合法的 YAML 文件。这是因为 YAML 也允许字符串被引号引起来，也将 `true` 和 `false` 都视为合法的布尔型，并且还支持与 JSON 数组和对象语法相同的内联列表与内联字典。但是千万别把你的 playbook 写成 JSON 版本，YAML 的亮点就是更易于人来阅读。

play

通过观察 YAML 或者 JSON 任意一种描述，可以清楚地发现 playbook 其实就是一个字典组成的列表。明确地讲，一个 playbook 就是一组 *play* 组成的列表。

这是我们范例中的 play[注6]：

```
- name: Configure webserver with nginx
  hosts: webservers
  sudo: True
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes

33  - name: copy nginx config file
      copy: src=files/nginx.conf dest=/etc/nginx/sites-available/default

    - name: enable configuration
```

注 6：　实际上，这是只包含一个 play 的列表。

```
    file: >
      dest=/etc/nginx/sites-enabled/default
      src=/etc/nginx/sites-available/default
      state=link

  - name: copy index.html
    template: src=templates/index.html.j2
              dest=/usr/share/nginx/html/index.html mode=0644

  - name: restart nginx
    service: name=nginx state=restarted
```

每个 play 必须包含下面两项。

- *host*：需要配置的一组主机。
- *task*：需要在这些主机上执行的任务。

play 就可以想象为连接到主机（host）上执行任务（task）的事物。

除了需要指定 host 和 task 之外，play 还支持一些可选配置。我们将稍后讨论这些配置，这里先介绍三个最为常见的配置。

`name`

一段注释，用来描述这个 play 是做什么的。Ansible 将会在 play 开始运行的时候将这段文字打印出来。

`sudo`

如果为真，Ansible 会在运行每个 task 的时候都使用 sudo 命令切换为（默认）root 用户。在管理 Ubuntu 服务器的时候这个配置会非常有用，因为 Ubuntu 默认不允许使用 root 用户进行 SSH 登录。

`vars`

变量与其值组成的列表。我们将会在本章中看到它的实例。

task

我们的范例 playbook 包含一个具有 5 个 task 的 play。下面是这个 play 的第一个 task：

```
  - name: install nginx
    apt: name=nginx update_cache=yes
```

name 是可选的，所以像下面这样编写 task 是完全合法的：

```
  - apt: name=nginx update_cache=yes
```

尽管 name 是可选的，我还是建议你配置它们。因为它们对于 task 的目的有非常好的提示作用。（name 配置对于其他人理解你的 playbook 非常有帮助。嗯，这个“其他人”也可能是几个月后的你自己。）就像我们已经看到过的，Ansible 会在 task 运行的时候将对应的 name 打印出来。最后，在第 14 章我们将会看到，可以使用 `--start-at-task <task name>` 参数告诉 ansible-playbook 从中间的 task 开始运行 playbook。当然，这需要你使用 name 来引用 task。

34 每个 task 必须包含由模块的名字组成的 key，以及由传到模块的参数组成的 value。在前面的范例中，模块名是 apt，参数是 `name=nginx update_cache=yes`。

这些参数告诉 apt 模块安装一个名为 *nginx* 的软件包，并在安装软件之前更新软件包缓存（与执行 `apt-get update` 命令等价）。

有一点非常重要：从 Ansible 前端所使用的 YAML 解释器角度来看，参数将被按照字符串处理，而不是字典。这意味着如果你想将参数分割为多行的话，你需要像如下这样使用折行语法：

```
- name: install nginx
  apt: >
      name=nginx
      update_cache=yes
```

Ansible 也支持能够将模块参数指定为 YAML 字典的 task 语法。当使用复杂参数的模块时，这个功能非常好用。我们将在第 102 页中的“Task 中的复杂参数：稍微跑个题”中讨论这种语法。

Ansible 还支持一个旧版本语法：它使用 action 作为 key，将模块的名字也放到 value 中。按照这个语法，前面的范例可以写成这样：

```
- name: install nginx
  action: apt name=nginx update_cache=yes
```

模块

模块是由 Ansible 包装后在主机上执行一系列操作的脚本[注 7]。这个描述看起来超级普通，但实际上 Ansible 模块千差万别。本章我们使用到的模块如下：

apt
: 使用 apt 包管理工具安装或删除软件包。

copy
: 将一个文件从本地复制到主机上。

注 7： 随 Ansible 一起发布的模块都是由 Python 编写的，但实际上模块可以使用任何语言来编写。

file
: 设置文件、符号链接或者目录的属性。

service
: 启动、停止或者重启一个服务。

template
: 从模板生成一个文件并复制到主机上。

查看 Ansible 模块文档

Ansible 随附的 `ansible-doc` 命令行工具可以用于查看模块的文档。可以将它看作 Ansible 模块的 man 手册。例如，想要查看 *service* 模块的文档，可以运行：

```
$ ansible-doc service
```

如果你使用 Mac OS X 操作系统，有个极好的文档查看器叫作 Dash（http://kapi.com/dash），它也支持 Ansible。Dash 为所有的 Ansible 模块创建了索引。Dash 是一个商业工具（在编写本书的时候售价为 19.99 美元），但是我发现它物超所值。

回顾一下第 1 章中，我们使用 Ansible 在主机上执行了一个 task，实现过程为：基于模块的名字和参数生成一个自定义脚本，然后将这个脚本复制到主机上并执行它。

Ansible 内置的模块有 200 多个，而且这个数字在每个更新的版本中都在增长。除此之外，你还可以在其他地方寻找第三方模块，或者自己编写模块。

将它们整合在一起

我们来总结一下，一个 playbook 包括一个或多个 play。一个 play 由 host 的无序集合与 task 的有序列表组成。每一个 task 仅由一个模块构成。

图 2-3 是一张实体关系图，它描述了 playbook、play、host、task 和模块（module）之间的关系。

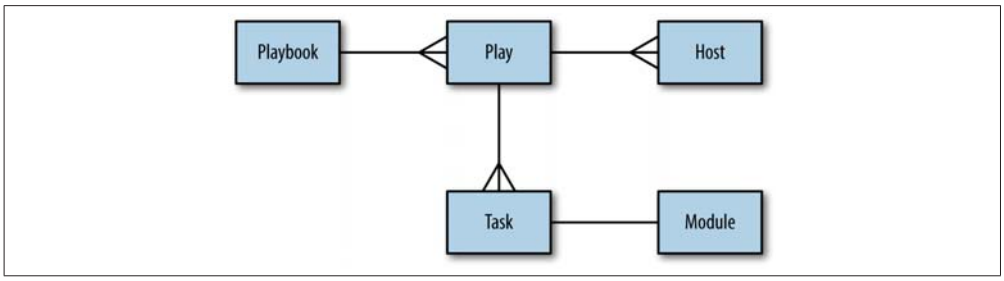


图2-3　实体关系图

36 执行 Ansible 后发生变化了吗？跟踪主机状态

当运行 ansible-playbook 时，Ansible 便会输出它在 play 中执行的每一个 task 的状态信息。

我们回头看看例 2-5，需要注意的是有些任务的状态显示 *changed*，而其他的状态都是 *ok*。例如，`install nginx` 任务的状态就是 *changed*，在我的终端上呈现为黄色。

```
TASK: [install nginx] ****************************************************
changed: [testserver]
```

而另一方面，`enable configuration` 任务的状态是 ok，在我的终端上呈现为绿色：

```
TASK: [enable configuration] *********************************************
ok: [testserver]
```

任何 Ansible 任务运行时都有可能以某种方式改变主机的状态。Ansible 模块会在采取任何行动之前先检查主机的状态是否需要改变。如果主机的状态与模块的参数相匹配，那么 Ansible 不会在这台主机上做任何操作，并直接响应 *ok*。

另一方面，如果主机的状态与模块的参数不同，那么 Ansible 将改变主机上的状态并返回 *changed*。

这个范例的输出表明，*install nginx* 任务的状态被改变了。这意味着在我运行这个 playbook 之前，这台主机上没有事先安装 *nginx* 软件包。而 *enanble configuration* 任务的状态没有被改变，这意味着这台主机上已经有了一个配置文件，并且与我希望复制过去的相同。这是因为我在 playbook 中使用的 *nginx.conf* 文件与 Ubuntu 的 *nginx* 软件包安装的 *nginx.conf* 一模一样。

本章稍后我们将会讨论到，Ansible 检测状态变化的机制可以通过使用 *handlers* 来触发额外的操作。但是，即使不考虑使用 handlers 方面，这仍然是一个可以用于反馈在 playbook 运行中是否被更改状态的好办法。

来点更酷炫的：添加 TLS 支持

让我们继续尝试一个更复杂的范例：我们将要修改一下之前的 playbook，以使得我们的 Web 服务器支持 TLS。这里我们要用到的新特性包括：

- 变量
- Handlers

TLS vs. SSL

在关于 Web 服务器的安全连接的相关术语中，你可能对 *SSL* 比对 *TLS* 更熟悉一些。SSL 是用于浏览器与 Web 服务器之间安全连接的旧版协议，而它已经被名为 TLS 的新版协议所取代。

尽管很多人继续使用术语 *SSL* 指代当前版本的安全协议，但是我在本书中会使用更准确的叫法：*TLS*。

例 2-8 所示为添加了 TLS 支持配置的 playbook。

例2-8　web-tls.yml

```
- name: Configure webserver with nginx and tls
  hosts: webservers
  sudo: True
  vars:
    key_file: /etc/nginx/ssl/nginx.key
    cert_file: /etc/nginx/ssl/nginx.crt
    conf_file: /etc/nginx/sites-available/default
    server_name: localhost
  tasks:
    - name: Install nginx
      apt: name=nginx update_cache=yes cache_valid_time=3600

    - name: create directories for ssl certificates
      file: path=/etc/nginx/ssl state=directory

    - name: copy TLS key
      copy: src=files/nginx.key dest={{ key_file }} owner=root mode=0600
      notify: restart nginx

    - name: copy TLS certificate
      copy: src=files/nginx.crt dest={{ cert_file }}
      notify: restart nginx

    - name: copy nginx config file
      template: src=templates/nginx.conf.j2 dest={{ conf_file }}
      notify: restart nginx

    - name: enable configuration
      file: dest=/etc/nginx/sites-enabled/default src={{ conf_file }} state=link
      notify: restart nginx

    - name: copy index.html
```

```
    template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html
          mode=0644
38  handlers:
    - name: restart nginx
      service: name=nginx state=restarted
```

生成 TLS 证书

我们需要手动生成 TLS 证书。在生产环境中，你需要从证书权威机构购买你的 TLS 证书。我们使用自签发证书主要是因为我们可以免费生成证书。

在你的 playbooks 目录下创建一个 files 子目录，然后在该目录下生成 TLS 证书和私钥：

```
$ mkdir files
$ openssl req -x509 -nodes -days 3650 -newkey rsa:2048 \
    -subj /CN=localhost \
    -keyout files/nginx.key -out files/nginx.crt
```

该命令会在 *files* 目录下生成文件 *nginx.key* 和 *nginx.crt*。证书的有效期是从创建之日起 10 年（3650 天）的时间。

变量

现在，在我们的 playbook 的 play 中多了一个名为 *vars* 的区段：

```
vars:
  key_file: /etc/nginx/ssl/nginx.key
  cert_file: /etc/nginx/ssl/nginx.crt
  conf_file: /etc/nginx/sites-available/default
  server_name: localhost
```

这部分中定义了四个变量并为每个变量进行了赋值。

在我们的范例中，所有变量的值都是字符串（例如：`/etc/nginx/ssl/nginx.crt`），但是任何合法的 YAML 对象都可以作为变量的值。除了字符串和布尔型之外，你还可以使用列表和字典。

变量不但可以在 tasks 中使用，而且可以在模板文件中使用。你可以使用 `{{braces}}` 的形式来引用变量。在 braces 的位置填写变量名，Ansible 将会使用变量的值替换它。

思考一下下面 playbook 中的 task：

```
- name: copy TLS key
  copy: src=files/nginx.key dest={{ key_file }} owner=root mode=0600
```

当 Ansible 执行这个 task 的时候，将会使用 `/etc/nginx/ssl/nginx/nginx.key` 替换 `{{ key_file }}`。

什么时候必须使用引号

如果你刚好在模块声明之后引用变量，YAML 解析器会将这个变量引用误解为内联字典。思考一下这个范例的运行结果：

```
- name: perform some task
  command: {{ myapp }} -a foo
```

Ansible 会尝试将 `{{ myapp }} –a foo` 的第一部分解析为字典而不是字符串，然后返回一个错误。这种情况下，你必须使用引号将参数引起来：

```
- name: perform some task
  command: "{{ myapp }} -a foo"
```

在你的参数中含有冒号的时候也会产生类似的问题。例如：

```
- name: show a debug message
  debug: msg="The debug module will print a message: neat, eh?"
```

msg参数中的冒号会误导YAML解析器。你需要使用引号将整个参数字符串引起来，以避免这个问题。

不幸的是，仅仅将参数字符串引起来并不能解决这个问题。

```
- name: show a debug message
  debug: "msg=The debug module will print a message: neat, eh?"
```

嗯，这下 YAML 解析器开心了，但是输出可不是你想要的

```
TASK: [show a debug message] ***********************************
ok: [localhost] => {
     "msg": "The"
}
```

debug模块的 *msg* 参数需要将字符串使用引号引起来以捕获空格。在这个特定的范例中，整个参数字符串和 *msg* 参数都需要使用引号引起来。好在 Ansible 同时支持单引号和双引号，所以你可以这样做：

```
- name: show a debug message
  debug: "msg='The debug module will print a message: neat, eh?'"
```

这样就会得到我们期望的输出了：

```
TASK: [show a debug message] ***********************************
ok: [localhost] => {
    "msg": "The debug module will print a message: neat, eh?"
}
```

如果你忘记在正确的地方使用引号，最终以一个非法的 YAML 文件执行，Ansible 会输出友好易读的错误信息。

40 生成 nginx 配置模板

如果你喜欢 Web 开发，那么你应该使用过模板系统来生成 HTML。不过有的读者可能没有使用过模板系统，所以在这里我还是举例介绍一下它的作用。模板就是一个文本文件，它具有一些特别的语法，可以指定变量，而这些变量最终会被替换成具体值。你可能收到过某些公司自动发送的邮件，那些邮件就很可能是由例 2-9 所示的邮件模板来生成的：

例2-9 一个邮件模板

```
Dear {{ name }},

You have {{ num_comments }} new comments on your blog: {{ blog_name }}.
```

Ansible 的使用场景并不是 HTML 或者邮件，而是配置文件。你肯定不想手动编辑配置文件，特别是你需要在多个配置文件中复用某些配置数据（例如，你队列服务器的 IP 地址或者数据库的授权信息）时，这种感觉就会更强烈了。更好的办法是：将这些具体到你的部署情况的信息记录在单一位置，然后通过模板生成所有需要这些信息的文件。

Ansible 的模板是使用 Jinja2 模板引擎来实现的。如果你以前使用过某个模板库，例如 Mustache、ERB 或者 Django 模板系统，你会感觉 Jinja2 似曾相识。

Nginx 的配置文件需要存放 TLS 证书和私钥的路径。我们将使用 Ansible 的模板功能来定义配置文件，以避免将那些可能经常变化的配置写死。

在你的 *playbooks* 目录下，创建一个名为 *templates* 的子目录，并如例 2-10 所示创建文件 *templates/nginx.conf.j2*：

例2-10 templates/nginx.conf.j2

```
server {
        listen 80 default_server;
        listen [::]:80 default_server ipv6only=on;

        listen 443 ssl;

        root /usr/share/nginx/html;
```

```
        index index.html index.htm;

        server_name {{ server_name }};
        ssl_certificate {{ cert_file }};
        ssl_certificate_key {{ key_file }};

        location / {
                try_files $uri $uri/ =404;
        }
}
```


我们使用 `.j2` 文件名后缀来表示这个文件是一个 Jinja2 模板。不过，你可以使用其他的文件名后缀，Ansible 不会在意，只要你喜欢就好。

在我们的模板里，我们引用了下面三个变量。

`server_name`
: Web 服务器的主机名（例如 `www.example.com`）。

`cert_file`
: TLS 证书的路径。

`key_file`
: 私钥文件的路径。

我们在 playbook 中定义了这些变量。

Ansible 在 playbook 中还使用 Jinja2 模板引擎对变量取值。回忆一下我们之前在 playbook 中看到的 `{{ conf_file }}` 语法。

早期版本的 Ansible 在 playbook 中使用 $（而非大括号）来引用变量。过去，你需要使用 `$foo` 来引用变量 *foo*，而现在则需要使用 `{{ foo }}`。`$` 已经被弃用，如果你在网上的 playbook 例子中看到它了，那么说明你在看旧版本的 Ansible 代码。

你可以在你的模板中使用所有 Jinja2 特性，但是我们没办法详细讨论所有这些特性。你可以查阅 Jinja2 模板设计者文档（*http://jinja2.pocoo.org/docs/dev/templates/*）获取详细的信息。其实你很可能并不需要那些高级的模板功能。你最有可能在 Ansible 中用到的 Jinja2 特性是 filter，我们将在后面的章节详细讨论它。

handler

让我们回过头继续来看 *web-tls.yml* 这个 playbook。这里有两个 playbook 元素我们还没

有讨论过。其一就是 handlers 区段，就像这样：

```
handlers:
- name: restart nginx
  service: name=nginx state=restarted
```


其二，有几个 task 中包含 notify 关键字。例如：

```
- name: copy TLS key
  copy: src=files/nginx.key dest={{ key_file }} owner=root mode=0600
  notify: restart nginx
```

handler 是 Ansible 提供的条件机制之一。handler 和 task 很相似，但是它只有在被 task 通知的时候才会运行。如果 Ansible 识别到 task 改变了系统的状态，task 就会触发通知机制。

task 将 handler 的名字作为参数传递，以此来通知 handler。前面的范例中，handler 的名字是 `restart nginx`。对于 nginx 服务器来说，我们需要在下列事件发生时重启它[注8]：

- TLS 密钥发生变化。
- TLS 证书发生变化。
- 配置文件发生变化。
- *sites-enabled* 目录下的内容发生变化。

我们在每一个与上述事件相关的任务中都声明一个 notify，以确保这些条件触发的时候 Ansible 会重启 nginx。

关于 handler 的几件需要牢记于心的事

handler 只会在所有任务执行完后执行。而且即使被通知了多次，它也只会执行一次。handler 按照 play 中定义的顺序执行，而不是被通知的顺序。

Ansible 官方文档提到 handler 唯一的常见用途就是重启服务和重启服务器。就我个人而言，我仅用它重启服务。尽管这只是一个相当小的优化，因为我们总是可以在 playbook 的末尾无条件地重启服务，而不是在变化时候触发它，并且重启服务通常并不需要很长时间，但是我仍然推荐使用 handler。

我曾经遇到的另一个关于 handler 的陷进就是，在调试 playbook 的时候，handler 可能会引起一些小麻烦。事情是这样的：

1．我运行了 playbook。

注 8：　我们也可以选择使用 state=reloaded 来重载配置文件，而不是重启服务。

2．我的一个 task 改变了状态并发起 *notify*。

3．后面的 task 发生了错误，Ansible 退出了。

4．我修复了 playbook 中的错误。

5．我再次运行 playbook。

6．重新开始执行后并没有 task 汇报状态改变，所以 Ansible 并不会执行 handler。

运行 playbook

像之前一样，我们使用 `ansible-playbook` 命令来运行 playbook。

```
$ ansible-playbook web-tls.yml
```

输出应该类似下面这样：

```
PLAY [Configure webserver with nginx and tls] **********************************

GATHERING FACTS ****************************************************************
ok: [testserver]

TASK: [Install nginx] **********************************************************
changed: [testserver]

TASK: [create directories for tls certificates] ********************************
changed: [testserver]

TASK: [copy TLS key] ***********************************************************
changed: [testserver]

TASK: [copy TLS certificate] ***************************************************
changed: [testserver]

TASK: [copy nginx config file] *************************************************
changed: [testserver]

TASK: [enable configuration] ***************************************************
ok: [testserver]

NOTIFIED: [restart nginx] ******************************************************
changed: [testserver]

PLAY RECAP *********************************************************************
testserver                 : ok=8    changed=6    unreachable=0    failed=0
```

打开你的浏览器访问 *https://localhost:8443*（不要忘记在 http 后面加上 s)。如果你像我一样使用 Chrome，你会看到一个非常吓人的类似“Your Connection is not Private”（你的

连接并不私密）这样的信息（见图 2-4）。

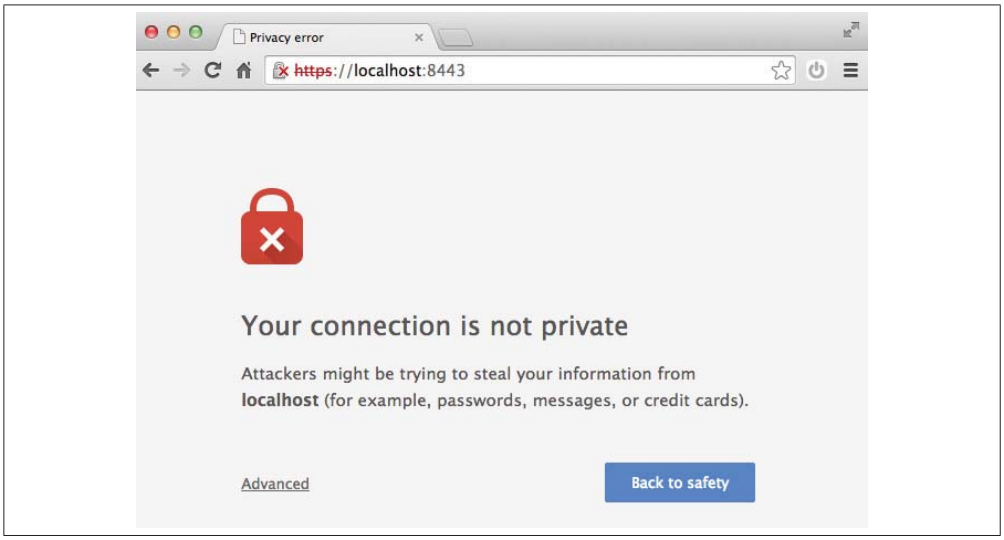


图2-4　以Chrome为代表的浏览器并不信任自签发的TLS证书

尽管报错信息很严重，但是别担心。这个错误是预料中的，因为我们生成了自签发的证书，而像 Chrome 这样的浏览器只信任正式的权威机构发行的证书。

在本章，我们讨论了很多关于 Ansible“做什么”的内容。这些内容描述了 Ansible 会在你的主机上做哪些事情。我们本节讨论的 handler 只是 Ansible 提供的流程控制机制之一。在后面的章节中，我们将会看到循环迭代和条件执行的范例。

在下一章，我们将要讨论关于“谁”的话题，具体来说就是如何定义与描述你希望 playbook 运行的目标主机。