网络专业人员书库

NetDevOps 入门与实践

余 欣 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

NetDevOps 入门与实践 / 余欣著 . —北京：机械工业出版社，2018.5
（网络专业人员书库）

ISBN 978-7-111-59909-8

I. N…　II. 余…　III. Linux 操作系统－程序设计　IV. TP316.85

中国版本图书馆 CIP 数据核字（2018）第 086384 号

NetDevOps 入门与实践

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：高婧雅　　责任校对：殷　虹
印　　刷：三河市宏图印务有限公司　　版　　次：2018 年 5 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：21.5
书　　号：ISBN 978-7-111-59909-8　　定　　价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

Praise 本书赞誉

（本书推荐人排名不分先后，以音序排列）

随着通信技术近年来颠覆性的发展和变化，对传统网络技术工程师们的挑战越来越大。作者以传统网工[⊖]成功转型的亲身经验撰写了本书，它直击传统网络工程师们的痛点，是难得的兼具实用价值和实践意义的“惊艳”之作，令人耳目一新！

——方芳，思科大中华区副总裁兼运营商 & 媒体广电事业技术部总经理

网络运维和系统运维本不是一个世界。技术栈、操作任务甚至运维价值观都是截然不同的，一直以来泾渭分明，各自精彩。

近年来虚拟网络的发展、SDN 的兴起，网络与 IT 系统逐渐开始跨界融合，而结合部分的故障定位、全局性的问题跟踪和优化成了传统运维的新盲区；云计算规模化的环境下，海量操作变更、复杂的关联定位，对传统人肉运维来说更是不可承受之痛。新的形势下，传统网络运维工程师的自我救赎之路，就是本书所倡导的 NetDevOps 理念：补齐 IT 系统技术栈，掌握必要的开发语言，熟悉主流的批量运维工具和基础服务，将自动化运维的理念延伸到网络领域，将研发的思维模式嵌入到传统的网络运维动作中，将网络运维标准化、自动化、智能化。

本书深入浅出展示了 NetDevOps 的理念、基础知识和最佳实践，值得有意转型的网络工程师深入研究学习。

——林恩华，中国移动苏州研发中心广州支持中心总经理助理

网络运维可视化、自动化和智能化的快速发展背后的本质诉求是能满足大型互联网公司的巨大网络规模增速和高效高质运维要求，具体又体现在人均运维效率和稳定性指标的极致追求上。每一位互联网企业的网络工程师都恰逢其时，有幸在网络运维领域引领技术发展的潮头并对各行各业中网络技术的发展产生一定的影响。网络运维 DevOps 就是网络工程师发展的方向，已在大型互联网公司深深扎根、蓬勃发展。余欣在阿里巴巴工作期间

⊖ 业界对网络工程师的简称，下文也会出现。

经历了网络工程师队伍转型的剧痛，并表现出了优秀的 DevOps 思路和能力。这本书作为网络 DevOps 入门指南写得深入浅出，非常符合网络 DevOps 的实际工作，各种细化的小场景、小步骤非常接地气，同时又富含 DevOps 的深层思想，我相信对传统网络工程师或初入行的网络工程师来说深具价值，推荐给大家研读学习。

——刘洋，阿里巴巴网络系统事业部总经理

伴随互联网业务的高速发展，网络规模持续快速增长，数量庞大的网络设备产生海量的运营数据，传统的人机交互的运维方式面临巨大的挑战。NetDevOps 利用 DevOps 的理念，推进网络运维的自动化与智能化，给网络运维带来了转机。本书介绍了 NetDevOps 产生的背景、发展历程，同时系统阐述了 NetDevOps 的框架体系、工具以及基本的软件编程知识，是国内难得的一本专业而又全面讲解 NetDevOps 技术的学习资料和参考手册，相信希望了解 NetDevOps 的网络同行们，能从本书中找到你们想要的内容。

——邵华，腾讯网络平台部网络架构中心总监

在软件工程领域中，DevOps 已经由一种文化演变成广泛落地的业务思维，将组织内的各个角色更紧密地联系在一起以提高生产力。但是在网络工程领域，受限于网络工程师技术栈及运维管理定势，如何理解 NetDevOps 思想进而在实际工作中更好地解决运维管理问题和新技术部署带来的挑战，仍存在不小的困难。

很高兴看到余欣用简明的语言和具体的场景将 NetDevOps 的方法论和实践进行了系统全面的呈现，是网络工程师、网络平台开发工程师不可错过的参考读物。

——宋磊，百度网络运维部技术经理

本书作者是网络行业的资深老兵，在 Cisco、Juniper 这样的网络设备制造商工作多年，也曾在阿里巴巴、京东金融的网络部门从事实际运维工作，拥有丰富的经验，亲身经历了 IP 网络的爆发式增长时代。面对最新的网络自动化运维的趋势，大量的传统运维工作必须转向软件自动化的方式，新的 SDN、NFV 等理念，也要求网络工程师具备软件编程能力。很多老网工在新的挑战面前，会有些眼花缭乱，不知从何入手。本书分享了作者自身的转型经验及丰富的实际案例，指出了一条切实可行的转型道路，对广大网工有非常好的参考价值，尤其是没有软件编程基础的网工。本书由浅入深地介绍了基本的概念和常用的工具，可以让大家少走弯路，节省很多自己去摸索试错的时间和精力。

——王卫，原瞻博网络大中国区总裁

由浅入深，有料清晰！作者结合自身在多家国际网络设备制造商和互联网公司的丰富经验，为读者指明了一条从传统向 NetDevOps 发展的转型之路。纯干货！值得一读！

——徐志骏，思科大中国华东区运营商事业部技术总监

本人与作者在 Brocade 共事期间，我们就意识到让老网工们快速转型 SDN 工程师是

不现实的，因为机器对机器的软件接口（API）不是网工们熟知的。找到一条有实战价值，门槛相对合理，容易启动的“工农结合”的路径就显得格外有吸引力。当前，作为一名新一代云网融合服务商的 CTO，团队建设的一个重要挑战和机会就是赋能老网工们，把建设运维实战经验与智慧总结形成清晰套路（算法），与专业码农们紧密配合，迅速实现运维排障经验软件化、自动化。与此同时，给网工们提供现实的发展演进路径，在实战项目中以商业价值目标为导向培养编程思维，接触机器接口，在一个个自动化的小任务中一步步实现自己的想法，获得真实成就感，成为新一代高度软件化的网络工程师、架构师和产品经理。针对这一目标，本书对 NetDevOps 相关的各个基础技术领域的功能、结构和过程维度的阐述简单直观而又高度实战。实验代码完整，注解清晰，实操容易上手，结果立竿见影。对数字化转型大潮中的网工们和相关技术团队的管理者们来说，本书不可不察。

——张宇峰，互联港湾 CTO

前 言 Preface

为什么要写这本书

清晨，我们做的第一件事是什么？睁开眼。睁开眼看手机里的朋友圈是否有更新，昨晚下的单是否已经安排送货，今天的天气是否依旧晴朗。而这些信息的更新都是通过互联网传递到你的手机上。在很多人眼里，手机有电而没有网络是一件非常痛苦的事情。互联网在中国的发展也就是 20 来年的事，但它已经渗透到了我们工作、学习和生活的方方面面。网络是新时代的基础设施，无论上面有多么丰富多彩的应用软件，它们都离不开网络。这些年，应用软件的迭代速度非常快。而网络在这几十年中却没有发生多大的变化（虽然网络带宽一直在指数级增长）。特别是网络工程师们日常的工作似乎还是和 10 年前甚至 20 年前一样。虽然，这几年 SDN（Software Defined Networks）在快速发展，但是物理网络仍然没有发生多大的变化。大量的网络工程师还是通过 Telnet 或 SSH 登录到网络设备上，然后一条一条地敲击各种各样的命令。应用软件越来越多，应用软件生命周期越来越短。这对网络提出了很多的挑战，网络工程师的工作压力也是直线上升。这几年随着上层应用 DevOps 思想的发展，网络自动化的需求也在不断提升。那些安分守己的传统网络工程师面临着转型的痛苦。

我是一个和网络打交道 20 来年的传统网络工程师，但我一直是一个不安分守己且会偷懒的人。早在我大学期间，为了和同寝室的同学一起玩一款叫“红色警戒”的游戏而接触了网络。从两台电脑之间使用串口互联进行对战，到使用同轴电缆后 8 个同学可以在一个地图中互相厮杀，再到 1999 年通过双绞线接入互联网。那个时候，几个寝室的双绞线都汇聚到了我们寝室，我不知不觉也成了 96 级化学系的网络管理员。日常的“工作”就是帮同学看看网络怎么不通了；谁的 IP 地址又和谁冲突了；如何从其他同学的电脑里复制一些电脑游戏等。活脱脱就是一个小型网吧的工作人员。随着 1999 年学校寝室接入了互联网，出于对“工作”的热情，我开始用 Linux 自己搭建一些服务，比如 DHCP、DNS、FTP、BBS 等。慢慢又干起了系统管理员的“工作”。

在千禧年（2000 年）的毕业季，我的第一份工作是在一家大型的纺织公司做系统管理

员和 DBA。这份工作和化学没有任何的关系。而日常的工作就是帮助新员工开账号，每天备份那些数据库的数据到磁带中。为了减少自己日常的工作就开始写一些自动化的脚本。其实，当时就是为了每天能偷点懒。开一个账号，懒得去点那么多次的鼠标。每天的备份任务，懒得去一个个地核对和比较，而是让脚本自己去核对，自己去比较，然后把检查后的结果发送 E-mail 给我。

2003 年考完 CCIE 后到一家为中国电信服务的系统集成公司。在这家公司有幸参与了中国电信 CN2（ChinaNet2）的建设工作。在网络建设的初期有大量的设备配置需要增加和修改。纯手工的操作让我觉得痛苦不堪，此时又萌生了“偷懒”的思想。我开始用 Python、Perl 等语言写了一些脚本用于设备配置的生成和修改。当时设备并没有丰富的 API 接口，大部分都是用 Telnet 模拟登录来操作设备。

2007 年我进入了 Juniper 工作，在这里接触了更多的网络自动化的内容，也写了很多自动化脚本来操作网络设备。比如，2008 年考完 JNCIE 后，有幸做了一年多的中国区 JNCIE 考官。JNCIE 的考官除了要发卷子外，还需要负责给考生判卷。也是为了“偷点懒”写了一些自己用的脚本提高判卷的效率。2009 年开始学习 JUNOScript（一种可以运行在 JUNOS 上的脚本语言），用 JUNOScript 来实现一些特殊的功能或者对命令进行重新格式化的输出。2010 年后由于需要经常参加设备的测试，开始使用 Python 等语言对 JUNOS 设备基于 NETCONF 协议进行操作。

2014 年到 2016 年，我先后在两家互联网公司做网络工程师，负责网络的规划与运维工作。由于互联网公司自身的产品迭代速度很快，对网络的适配性也提出了更多的需求。虽然在互联网公司有很多的程序员，但大部分的程序员对网络和网络设备的理解远逊于网络工程师。这就导致了网络自动化的开发工作比较难推进。因此，我结合自己的编程能力和对网络的理解开始用代码去实现网络自动化的任务。

从 2016 年到现在，我一直在 Cisco 工作。在这里我接触到了 DevNet（https://developer.cisco.com）。在 DevNet 的网站上我看到和学习了很多关于基础网络设备的编程知识。在 2016 年，Cisco 发布了思科全数字化网络架构（思科 DNA），这个平台不仅提供了实现全数字化的路线图，而且为网络工程师提供了网络自动化和网络安全的途径。这个平台的很多理念和架构为我写这本书提供了很多的帮助。

在这 20 来年的时间里，我积累了一些使用程序来操作网络设备的经验。一方面是想把这些经验分享给大家；另一方面也是想帮助那些想转型的传统网络工程师。这就是我写这本书的初衷。另外，我还想告诉广大的网络工程师们开发一个小工具用来管理设备其实并没有那么难。对于我这样一个非软件专业的人而言并没有觉得吃力，反而在开发中获得了更多的自信，也偷了“懒”。

最后，希望这本书能给广大的网络工程师在转型过程中带来一些帮助，也希望大家能少走弯路。

本书特色

首先，本书是专门针对网络工程师而写的。书中关于 Bash 和 Python 的基本语法部分使用了网络工程师更加熟悉的内容，并且提供了一些网络设备上的运行情况。

其次，本书的重点是如何编写和网络设备相关的代码。因此，在书中提供了很多关于如何处理网络设备输出的文本的例子，以及处理网络相关的数据。

最后，本书并不是一本纯粹讲解编程的书，而是一本从理论到实践的综合书籍。

读者对象

- ❑ 网络架构师
- ❑ 网络运维工程师
- ❑ 网络运维开发人员
- ❑ 网络与系统管理人员
- ❑ 网络规划与设计人员
- ❑ 网络专业在校学生

如何阅读本书

本书分为五篇，共计 14 章内容。

第一篇为概念篇，这一篇主要讲述什么是 NetDevOps，以及如何开始 NetDevOps 实践之路，包括如下 2 章内容。

第 1 章 从 SDN 开始谈起，讲解在 SDN 的大背景下，传统的网络都发生了什么变化，而这些变化给传统网络工程师带来了哪些影响。最后介绍了什么是 NetDevOps，NetDevOps 需要我们学习什么样的技能才能胜任。

第 2 章 在业务快速迭代的推动下，传统 IP 网络的自动化需求在不断增强。大量的网络工程师面临着新的挑战。这章介绍如何从零开始逐步过渡到 NetDevOps。这章将重点讲解 4 个话题：首先，在 NetDevOps 开始之前需要做什么；其次，在进行 NetDevOps 开发时，如何选择开发语言；再次，一些常见的 NetDevOps 开源工具或平台如何选择；最后，在进行 NetDevOps 时，对网络设备有哪些要求。

第二篇为基础篇，这一篇主要介绍了如何构建 NetDevOps 的工作环境以及在这些环境中的常用工具，包括如下 4 章内容。

第 3 章 介绍在 Linux 环境下，如何使用 Linux 下的工具登录网络设备，以及使用 SSH 工具建立一些 SSH 的隧道。

第 4 章 介绍在 Linux 环境下，如何使用一些工具获取网络设备的信息，以及获取网

络的可达信息，涵盖 SNMP、traceroute、ping 等工具。

第 5 章 使用 Linux 中三大文本处理利器（grep、awk 和 sed）来处理网络设备输出的文本内容。这些文本内容包括命令行的输出、设备的配置以及设备的日志信息等。这些工具可以帮助网络工程师快速地获取相关的数据和信息。

第 6 章 在 NetDevOps 的实践过程中，我们需要搭建一些基础的服务。这些服务包括 TFTP、DNS 和 DHCP 等。在微模块流行的时代，网络工程师使用 Docker 可以快速地构建起这些基础服务。

第三篇为提高篇，这一篇将开始介绍编程相关的内容。这一篇都是编程的一些基础知识，包括如下 3 章内容。

第 7 章 这一章主要介绍 Linux 环境或网络设备上的 Bash 编程基础知识。通过 Bash 基本语法并结合一些工具，我们可以和设备进行简单的交互或处理一些数据。

第 8 章 这一章主要介绍 Python 的编程知识。本书的大部分编程内容都是基于 Python 语言的。因此，这一章是后续章节的基础。这一章关于 Python 的基本语法是专门为网络工程师重新编写的。使用的例子将是网络工程师比较熟悉的内容。

第 9 章 我们在和网络设备进行交互或者进行网络相关的编程时，经常需要处理一些常用的数据类型，这些数据类型包括 JSON、XML、YAML 和 YANG。熟练掌握这些数据类型的处理是编程的基础。在这章，我们将介绍上述这四种数据类型的常用处理方法。

第四篇为实践篇，这一篇将通过一些实际的例子来介绍，包括如下 3 章内容。

第 10 章 NetDevOps 必然需要和网络设备进行交互，从而获得我们需要的数据。本章将介绍三种常见的连接网络设备的方法，它们分别是：命令行登录、NETCONF 以及 REST。

第 11 章 连接到网络设备后就可以获取很多的信息，其中通过命令行获取的数据大部分是半结构化的数据。这些半结构化的数据需要进行结构化处理。这一章将通过几个 Python 的模块来处理这些数据。

第 12 章 我们在处理网络相关数据时，有两种常见且特殊的数据需要处理，它们分别是网络地址和网络拓扑数据。同样，我们将通过几个 Python 的模块来处理这些数据。

第五部分为案例篇，这一篇将介绍 3 个常见的案例来帮助大家更好地了解和掌握 NetDevOps 的相关内容，包括如下两章内容。

第 13 章 众所周知，绝大多数的网络设备都会有配置文件，获取和管理这些配置文件是 NetDevOps 工作的基础。通过程序化的方式自动地获取这些配置就打通了程序和网络设备之间的通道，这是后续获取更多信息的基础。另外，网络设备的配置文件也是最需要且被优先管理的内容，这些内容的版本管理也是非常重要的。本章将通过网络设备的配置管理案例来描述如何多厂家、并发地与网络设备进行数据交互。

第 14 章 网络运维与管理的独特之处是，该工作是基于网络拓扑的。获取和处理网络拓扑是基本功能。该章通过两个小的案例来介绍，它们分别是：基于 ISIS 协议来获取网

络拓扑并进行简单的网络拓扑分析；使用 BGP 协议进行简单的网络流量调度。

其中，本书的第 2 章、第 8 章、第 9 章、第 10 章、第 11 章是重点。如果你有一定的 Python 编程基础，那么可以参考第 9 章及之后的章节，这些章节提供了 Python 用于网络管理与维护常用的一些模块，这些模块可以提高你的工作效率。如果你是一位传统的网络工程师且对编程和 Linux 环境不是十分了解，请从本书的开头读起。笔者希望通过本书的内容能循序渐进地带领大家走上 NetDevOps 之路。

勘误和支持

由于笔者的水平有限，编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。如果读者朋友有更多的宝贵意见，欢迎你发送邮件到 netdevops@hotmail.com 联系我。本书的大部分代码示例都放在 GitHub 上，其地址为 https://github.com/netdevops-engineer/newbie_book。期待能够得到大家的真挚反馈，在技术之路上互勉共进。

特别致谢

这里要特别感谢毛厚君先生，他是这本书的第一位读者，不但给了我很多的宝贵建议，而且帮我润色了全书的文字。如果没有他的帮助我想是很难完成这本书的。

致谢

在本书的写作过程中得到了很多同事和朋友的支持与帮助。没有你们的支持与帮助，本书将难以如期完成。

在本书的写作过程中需要实验环境，感谢徐晓东先生为我提供了便利。

感谢思科同事们的支持和鼓励，他们是方芳女士、徐志骏先生、杨骏先生、刘佳女士等。

感谢身在美国的朋友杨文嘉先生提供了关于 Arista 产品和技术的相关信息。

最后，我要特别感谢我的家人，我为写作这本书，牺牲了很多陪伴他们的时间，但也正因为有了他们的付出与支持，我才能坚持写下去。

谨以此书献给我最亲爱的人，以及众多的网络工程师朋友们！

余　欣

目　录 Contents

第一篇 Part 1

概 念 篇

欢迎来到《NetDevOps入门与实践》。

在第1章，我们将讨论一下NetDevOps的一些基本概念以及NetDevOps与SDN（软件定义网络）相关的一些内容，并且介绍一下为什么需要NetDevOps。

第2章主要涉及如何开始NetDevOps。随着上层业务的推动，传统的IP网络面临着前所未有的挑战，网络的自动化需求在不断增强，如何从零开始逐步向NetDevOps过渡，这是目前大量传统网络工程师的困惑所在。本章将涉及以下几个话题：

- 开始NetDevOps之前首先需要做什么；
- 关于NetDevOps语言的选择问题；
- 一些常见的NetDevOps开源工具的选择；
- 网络设备需要具备什么能力以及如何来管理它们。

Chapter 1 第 1 章

NetDevOps 理念与要义

自从 20 世纪八九十年代 Internet（互联网）出现以来，IP 网络在全球出现了爆发式发展。近几年来，随着大量计算机网络应用，特别是云计算、大数据以及互联网 + 在各行各业的渗透，网络变得无处不在。虽然网络不是万能的，但是现代社会没有网络几乎已经变成万万不能了。各种各样纷繁复杂的应用都承载在 IP 网络之上，网络规模变得越来越大，网络结构也变得越来越复杂。这些变化给网络工程师带来的压力在逐年增加。如何提高网络运维的效率、提升网络操作准确性以及网络业务可用性是网络工程师一直在苦苦探索的方向。

对于 NetDevOps，你需要向广大网络工程师、网络规划人员以及大部分运维平台开发工程师解释清楚它到底是什么、它包含哪些内容，以及它和这几年火热的 SDN（软件定义网络）有什么关系。

1.1 从 SDN 开始说起

在这几年的 IT 圈子中，最火的名词有云计算、大数据以及人工智能（AI）。在网络，特别是基础网络圈子里，最火的也许就是 SDN 与 NFV 了。我们在全球 IT 相关的大会中都可以看到它们的身影。在国内，网络厂商的技术交流会，以及互联网公司、运营商、企业的技术分享会都会提到 SDN 和 NFV 相关的解决方案。在网络运营与维护工程师聚集的论坛里也经常看到它们的主题。比如，NANOG（北美网络运维论坛 https://www.nanog.org）也有 SDN 与 NFV 相关的内容。作为在网络行业中工作了近 20 年的网络工程师，笔者和很多在这个行业中的网络工程师们一样，对于 SDN，也许一开始是拒绝的。但是随着 SDN 的声音越来越大，参与的人也越来越多，很多像笔者一样的网络工程师也开始关注和参与到这个领域。

那么 SDN 究竟和 NetDevOps 有什么关系或关联呢？正是 SDN 的出现和发展，推动了传统网络设备管理维护方式的变革：一方面，SDN 让网络工程师不再只依赖 CLI 命令行（部分 Web）方式对设备进行逐台的操作管理，毕竟逐台通过 CLI 的管理方式是一种非常低效且容易出错的运维方式；另一方面，SDN 推动了传统网络设备厂商自身的变革，各厂商开放出更多的 API 接口，供用户侧进行自动化的编排和调用。自动化这几年的呼声越来越大，自动化上线、自动化部署、自动化运维、自动化修复，甚至还可以和业务进一步结合。实际上，NetDevOps 追求的目标就是网络资源的接口化、自动化以及智能化。借助于 NetDevOps，我们可以提高运维的工作效率，提升业务部署的准确性，提供网络资源的可编程性。

在正式进入 NetDevOps 之前，我们先来了解一下 SDN 的发展状况。无论是 SDN 具体的技术细节，还是 SDN 的建设思想，SDN 对全球的学术研究机构、标准化组织以及设备厂商都已经产生了深远的影响。了解 SDN 技术的概貌及其相关的实现架构，有助于我们理解 NetDevOps。

1.1.1　OpenFlow 打开了新的一扇窗

对于 OpenFlow，我想大家应该并不陌生。有人认为 OpenFlow 是 SDN 的“Hello World”，也有人觉得它给网络带来了新的活力。众所周知，Martin Casado 在斯坦福大学读博士期间领导并开发了 OpenFlow 协议，当时他的博士导师是 Nick McKeown。这个协议不仅定义了网络设备数据转发平面的内容，同时也定义了控制平面的内容。简单来说，通过 OpenFlow，可以把控制平面从硬件中剥离出来，可以开发出更加智能、更加集中的控制中心。而转发平面把原来网络设备 ASIC 等芯片进行了更加高级的抽象化处理，让开发人员不用深入了解硬件底层的处理方式，就能够获得硬件带来的性能。《OF-CONFIG 1.2 ONF TS-0161》[⊖] 白皮书给出 OpenFlow 高度抽象的逻辑图（见图 1-1）。从这个图中，我们可以清晰地看出：OpenFlow 协议是 OpenFlow 控制器与 OpenFlow 交换机之间通信的协议，而交换机的其他操作使用了 OF-Config 这个接口协议。如果我们把这个结构映射到传统的路由器或者交换机中，OpenFlow 协议可以类比为传统设备的 RIB（Routing Information Base，路由信息表）与 FIB（Forwarding Information Base，转发信息表）之间的协议。

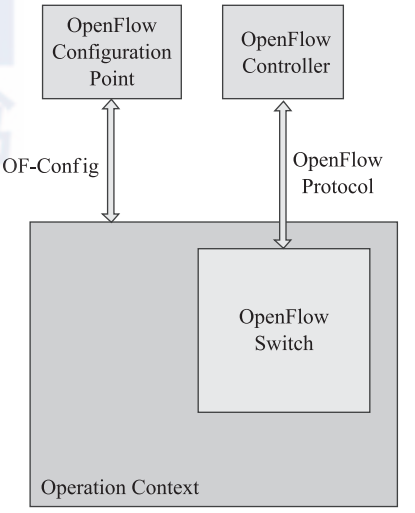


图 1-1　OpenFlow 逻辑图

⊖ https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf。

对于传统的网络设备，这里的通信协议往往是私有的，并没有开放给普通的使用者；而 OpenFlow 协议则定义了这部分内容。除了 OpenFlow 协议，OF-Config 定义的是如何管理和运维网络设备。OF-Config 定义了机器与机器之间的交互方式。它和设备之间的通信使用 NETCONF 作为底层的通信协议，使用了大量的 YANG 模型对数据结构进行了定义，并且给出了 UML（Unified Modeling Language，统一建模语言）的结构。

OpenFlow 协议本身并不是一场变革，它所描述的方式、方法不是传统网络工程师所熟悉的领域，反而是软件工程师所熟悉的领域，它从另外一个视角来描述网络的管理与维护工作。它让传统的人机交互命令演变为控制器与机器之间的交互，从而可以实现集中化、统一化和程序化的网络管理维护方式。

NetDevOps 的初衷也是希望通过机器与机器之间的交互来实现更加高效、更加准确的网络管理，而不只限于对网络设备的管理；另外，也希望通过对网络资源进行二次封装，从而为上层应用提供简单灵活的编程接口。

1.1.2 简单聊聊 SDN 控制器

接着前面谈到的 OpenFlow 相关的一些内容。OpenFlow 自身并没有定义和解决控制平面的问题。在 OpenFlow 应用场景中，我们通过 SDN 控制器来保证运行 OpenFlow 的网络能正常地运作。SDN 控制器完成的是控制平面的工作，它是通过软件来实现的。为了满足不同场景的应用需求，控制器的内容会更加抽象化，在表述控制器功能的时候也更加软件工程化。在 SDN 网络（包括 OpenFlow 网络）中，和智能相关的内容大部分由 SDN 控制器（OpenFlow 网络对应的控制器通常称为 OpenFlow 控制器）完成和实现。“控制平面与转发平面分离”是 SDN 网络与传统网络的主要区别。转发平面常常被定义为毫无智能可言的“傻终端”，完全根据控制平面的指令来进行转发。

在开源的世界里，目前比较主流的 SDN 控制器如下：

- ❑ OpenDayLight（https://github.com/opendaylight）；
- ❑ ONOS（https://github.com/opennetworkinglab/onos）；
- ❑ Ryu（https://github.com/osrg/ryu）；
- ❑ Floodlight（https://github.com/floodlight/floodlight）。

SDN 控制器软件有很多，上面列出的只是其中很少的一部分。SDN 并不是本书的重点，因此也不会在这里详细的展开，有兴趣的读者可以找相关的书籍或者文章进行了解。

在这里，我们以 OpenDayLight（简称 ODL）为例子简单地看一下 SDN 控制器的框架设计图（https://www.opendaylight.org/odlboron 网站提供了更加清晰的图片）。

从 OpenDayLight 硼版本框架结构图（见图 1-2）中，可以清晰地看出以下几点。

首先，最下方是网络设备，既包括具备 OpenFlow 功能的设备，也包括 Open vSwith 这样纯软件的设备，还包括了一些其他的硬件物理设备。

其次，ODL 的南向接口（即控制器与网元设备通信的接口）不单单只有 OpenFlow 与

OF-Config，还包括 NETCONF、BGP、PCEP、SNMP 等多种接口。ODL 实现了基于南向接口的网络服务抽象层和基于抽象服务的接口转换。通过这一层的转换，能够让网络设备提供的描述方式和能力转化为面向业务的描述方式及接口。这就方便了应用层的开发人员进行网络相关应用程序的开发，他们并不需要深入了解网络底层的内容。

最后，控制器提供了北向 API 接口（有 RESTful、RESTCONF、NETCONF 等接口类型），这是应用开发人员编程所使用的接口。其他更加上层的系统或者平台可以通过调用这些接口来完成其开发，完成应用对网络的调度和管理。ODL 提供了 GUI 的开发框架，应用开发人员也可以基于这个开发框架来开发应用程序，并实现对网络的管理和调用。

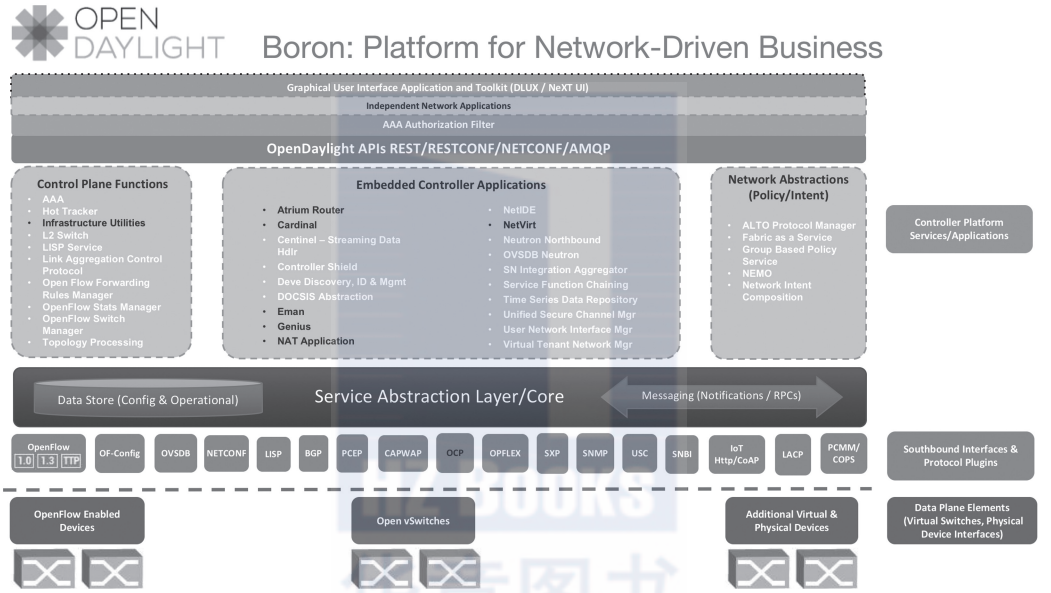


图 1-2　OpenDayLight 硼版本框架结构图

总之，SDN 控制器软件是应用开发人员和网络设备（无论是硬件的还是软件的）之间的桥梁。通过 SDN 控制器，应用开发人员可以更多地专注于业务逻辑，从而开发与网络相关的应用程序。因此，在 SDN 控制器设计的时候，无论是南向接口还是北向接口都会更加关注机器与机器之间的接口。这也许是很多的传统网络工程师感觉 SDN 控制器比较难学习和难以掌握的地方。正像前面提到的，网络工程师擅长的还是人与机器交互的界面，这也导致了网络工程师在接触 SDN 控制器时，通常也只是关注 SDN 控制器所提供的 Web UI 功能的原因。这样的思路需要转变，无论是 SDN 还是 NetDevOps，都应该更加关注机器和机器之间的接口部分。

1.1.3　NFV

随着 SDN（软件定义网络）的发展，NFV（Network Functions Virtualization，网络功能

虚拟化）在 SDN 领域也越来越多地被提及。NFV 是 ETSI（European Telecommunications Standards Institute，欧洲电信标准协会）在 2012 年 10 月发布的白皮书㊀中定义的。但是，NFV 这样的产品形态并不是 2012 年后才有。例如，2006 年春季推出的 Vyatta OFR㊁是一个软件的路由器，它可以运行在很多的虚拟化的平台上。又如，2009 年，Cisco 推出 Nexux 1000v㊂的虚拟交换机产品，它可以运行在 VMware 等虚拟化平台上，提供软件交换机的功能。另外，防火墙、负载均衡等领域都有虚拟化软件的产品，如 Juniper 的 vSRX㊃虚拟化和 cSRX㊄容器化的软件防火墙产品。ETSI 除了概念化了 NFV，还定义了 NFVI（Network Functions Virtualization Infrastructure，网络功能虚拟化的基础设施）以及 NFV-MANO（Network Functions Virtualization Management and Orchestration，网络功能虚拟化的管理与编排）等框架。这些框架为大规模、自动化地部署 NFV 提供了很好的指导思路与建议。

NFV 让网络服务功能的形态发生了不小的变化。现在物理服务器的性能越来越好，单台物理服务器上的虚拟机或容器就可以组成一定规模的小型网络。 根据现在服务器的性能及应用的要求，一台物理服务器通常可以虚拟化出 10～100 台虚拟机。而每台虚拟机又可以有 10～100 甚至更多的容器。这样，在一台物理服务器上就可以产生 100～10 000 个服务节点。这个规模已经超过了很多小型园区网的规模。物理服务器内部的网络结构也正变得越来越复杂。

在 NFV 的环境中，为了让服务节点之间的网络更加灵活，引入了 VxLAN（Virtual eXtensible Local Area Network）等技术。VxLAN 通常应用在 Overlay 的网络环境中，即在原有的三层网络（IP 网络）上又衍生出二层网络，类似传统网络中用 GRE 隧道方式组成的网络。这里我们不讨论 GRE（或 NVGRE）和 VxLAN 两者之间的区别，这并不是本书的重点。Overlay 网络中存在着大量非状态的隧道，并且这些隧道的生命周期往往是非常短的。这是因为按需服务的理念使得服务节点虚拟机或容器的生命周期正变得越来越短。

网络功能的软件化与虚拟化使得网络功能部署更加灵活和快捷，这势必会促使 NFV 软件大规模发展和应用，软件网元数量将会大大超过现在物理网元设备。同时，业务需求的频繁调整和变化，将使得 Overlay 网络拓扑的生命周期越来越短。管理和维护由软件网元组成的网络将是一个很大的挑战。

1.1.4 云和 SDN

“云”应该是大家再熟悉不过的概念了。现在有很多的公有云平台，如美国亚马逊的

㊀ https://portal.etsi.org/NFV/NFV_White_Paper.pdf。

㊁ https://wiki.vyos.net/wiki/Vyatta。

㊂ https://communities.cisco.com/community/technology/datacenter/data-center-networking/nexus1000v/blog/2009/7。

㊃ https://www.juniper.net/us/en/products-services/security/srx-series/vsrx。

㊄ https://www.juniper.net/us/en/products-services/security/srx-series/csrx。

AWS 云平台、微软的 Azure 云平台以及谷歌的云平台等；国内有阿里巴巴的阿里云平台、腾讯的云平台、百度的云平台以及青云等。另外，大量企业使用了 OpenStack、VMware 等搭建企业的私有云或公有云平台。在私有云和公有云之间又存在很多混合云的环境。从网络的视角来看，这些云的通信可以分为两大类：一类是云内部的通信。这通常是在一个 DC（Data Center，数据中心）内部的通信。另一类是云与云之间的通信。这里有些是私有云之间的通信，有些是公有云平台之间的通信，有些是公有云与私有云之间的通信。这些云之间的通信有些是专门的线路互联，例如，亚马逊的 AWS 平台就提供直接网络互联的服务。有些是没有专门的链路进行连接的，它们之间通过现在的互联网进行互联。

当计算、存储等资源云化后，这些资源的弹性变大了。换个角度来看，这些资源池所提供的服务节点虚拟机的生命周期也变短了，需要时启用，不需要时释放。这些资源都是网络中的节点，云越多，可用的资源也会越多，并且所有这些资源都会愈加依赖网络进行通信。散布在各地的云资源需要通过网络进行动态的组合，这势必会迫使网络具备动态的调整能力。连接云节点与云节点之间的网络无论是通过底层的物理网络来实现，还是通过上层的软件网络来实现，都需要具有一个供机器管理调用的接口，才有可能实现对网络的监控、创建、修改以及撤销等操作功能。

在网络中传送的数据最终还是会落到物理连接上。因此，物理网络（有时候也把它称为传统网络）同样需要实现更多机器与机器的通信接口。正是像云业务这样上层应用的不断“推动”，才使得网络的接口和自动化能力需要不断提升，从而达到更高的业务敏捷性。同时，软件定义网络技术的发展“拉动”了网络自身的变革。我们可以清晰地看到，网络的变革体现在以下几个方面。

首先，网络管理接口数据结构化，这里比较有代表性的是 NETCONF 协议的定义以及 YANG 模型对网络设备的配置与信息输出结果的结构化定义。关于 NETCONF 协议与 YANG 模型这两部分的内容，在后续的章节中会进行比较详细的介绍。

其次，在 SDN（软件定义网络）中强调转发与控制分离的架构。这种架构实际上是让硬件与软件进行了分离。软件部分的控制器在设计上提供了非常丰富的北向接口。而这些北向接口的定义，都是为了实现机器与机器（程序与程序）之间的通信。当网络设备的 API 越来越丰富时，在这样的环境中进行自动化运维就不是难事了。

大家也许注意到，刚才一直在说机器与机器通信的问题。正是类似像“云”这样的业务和 SDN 这样的技术促进了网络设备接口的升级。无论是商业公司的产品还是开源的项目，均提供了比以前更多的机器与机器交互的接口，而不再只是关注人与机器交互的接口。较典型的人与机器交互的接口是 CLI（Command Line Interface，命令行）以及 Web 界面。而这样的接口是很难与另外一台机器进行交互的。只有让更多的机器参与到业务流程中，才能提供更快的效率。网络提供的服务在业务平台的下层，如果网络能够给上层业务应用提供更多的可供机器使用的接口，网络被使用的效率自然也会得到快速提高。

1.1.5 SD-WAN

SDN 一直被认为缺少理想的应用场景。而自 2014 年开始出现的 SD-WAN（Software Defined-WAN）正逐渐成为 SDN 最热门的一种应用场景。截至 2017 年，业界对 SD-WAN 的定义还存在着一些分歧，具体可以参考 Wikipedia（https://en.wikipedia.org/wiki/SD-WAN）的内容。

在笔者看来，网络从功能、物理规模和特点方面来区分大致可以分为以下三大类。

1）**广域网**（Wide Area Network，WAN）。Wikipedia 给出的定义如下：广域网是指连接不同地区局域网或城域网计算机通信的远程网。广域网通常跨接很大的物理范围，所覆盖的范围从几十千米到几千千米。由于长途链路的成本是这张网络的主要成本，因此这张网络的拓扑结构差异性会比较大。广域网存在链路按需使用的需求，例如早期的 ISDN 链路就是按时长进行计费的；也存在网络流量调度的需求，即合理地利用现有的链路。Google B4 在这个领域做了很多的尝试[⊖]。

2）**园区网络**（也称为局域网）。相对于广域网而言，它的覆盖范围通常会比较小，常常仅限于一幢或者几幢楼的内部，并且其带宽相对广域网往往会大很多。园区网的主要功能是提供大量的信息点并管理这些信息点接入的用户，这些信息点可以是一些传感器或者是人机交互的终端。其主要特点是要应对各种各样的接入方式以及安全接入的需求。园区网的网络拓扑相对比较固定，通常是双星形的拓扑结构。

3）**数据中心网络**（Data Center Network，DCN）。这个网络的物理范围一般会更小，只覆盖一个或数个较近的机房。这个网络的拓扑非常标准化，目前最为流行的设计是 SPINE-LEAF 的结构[⊜]，设计完成后几乎不会有大的变动。其主要的特点是为服务器和服务器之间的通信提供高速的带宽，通常也称之为横向流量带宽。

而 SD-WAN 首先要解决广域网的互联互通，其次需要满足网络的快速开通与灵活部署，另外还要解决一些广域网优化的问题。SD-WAN 会大量引入 NFV，NFV 的灵活性在 1.1.3 节中有介绍。如何快速部署业务、如何管理网络节点、如何检测广域网的通信质量、如何优化广域网的转发路径、如何运营广域网，这些都是 SD-WAN 需要解决的问题。作为软件定义的广域网，SD-WAN 必然会使用很多编排系统，这些编排系统也必然会大量用到网络设备的 API。对于这样的网络，采用 DevOps 的运维方式和业务部署将是一个不错的选择。

1.2 NetDevOps，你需要知道的事

1.2.1 什么是 NetDevOps

NetDevOps 是网络（Networks）、开发（Developments）与运维（Operations）三个单词

⊖ https://conferences.sigcomm.org/sigcomm/2013/papers/sigcomm/p3.pdf。

⊜ 可以参考 http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/white-paper-c11-737022.html。

的复合词。很显然，这个单词已经非常清晰地表达了其所包含的内容：网络的运维工作需要开发者来一起参与进行，通过程序化的代码来完成大量运维的工作。推动这种变化（或称为演进）有来自多方面的因素：网络规模变大带来维护工作量的增加、应用快速上线、快速响应需求、自动化运维思想的驱动等，总之就是需要提高运维效率，给运维人员降低工作负荷，提升网络操作的准确性，降低误操作。从定义来看，NetDevOps 包含很多方面的内容，它既包含了技术的一些细节，也涵盖了很多非技术的内容。实际上，NetDevOps 代表了一种思路、一种方法论、一种与时俱进符合当前业务发展需要的新型的网络管理和运维手段。

本书主要从技术入手，侧重介绍实践过程中能直接应用的工具与方法，在最后会给出几个常见的应用场景的实际案例。

NetDevOps 不一定都要通过编程来实现。毕竟对于绝大多数网络工程师而言，编程属于陌生或跨界的领域，要求传统网络工程师一下子就能编写出大量代码来实现自动化的网管和运维也是比较困难的。学习和实践 NetDevOps 并不一定需要我们从头开始写每一行代码，我们可以学习和掌握一些已有的工具和现成的程序库来实现我们的想法、达到我们的目的。尤其是对于初学者而言，有现成 NetDevOps 工具能实现的就尽量不要使用编程，有现成的语言平台库能实现的功能就尽量不要去一行一行地去撰写代码。可以逐步编写更多的代码或基于已有的代码模板进行修改，这是一个循序渐进的过程，只有你一路坚持下来才能体会出这其中美妙的、得心应手的滋味。

1.2.2　NetDevOps 适用环境

现在得益于 IT 技术的发展，很多行业自身的自动化程度越来越高。随着人工智能的逐步普及，以后机器将会能做更多的事情。但是目前很多的网络规划与运维工程师们，特别是在国内，大家都还不太善于利用机器来帮助自己做更多的事情。在云计算的大趋势下，无论是网络的自动开通，还是网络的变更运维，都需要非常高的自动化程度。目前不论是传统的网络还是 SDN 相关的网络，都适合采用 NetDevOps 的方式进行运维管理。

在传统的网络中，设想某网络工程师管理维护了一套企业园区网，假设这个园区网从接入到汇聚再到核心和网络出口一共拥有 100 台网络设备。在日常的管理维护方面，网络工程师需要监控这些设备的运行状态，并定期做设备配置的备份或设备版本的升级；当然也会经常有新业务上线，对应网络及安全的策略变更等工作。可想而知，如果没有类似 NetDevOps 工具的帮助，此网络工程师的工作量将会非常庞大。而如果能够借助 NetDevOps 来完成此类日常事务，那么此网络工程师的工作量可以缩减为原来的十分之一，甚至更少。由机器完成此类重复类（循环类）的信息统计，其效能将万倍于人的手工操作。

在云服务商自身的 SDN 网络中，NetDevOps 已经得到较为广泛的使用，包括公司层面 DevOps 的开发（此类 NetDevOps 通常会跟公司主营业务应用以及服务器虚拟化平台进行融

合，打通业务自动化链条上的各个环节）以及网络运维部门自身的维护 NetDevOps 工具的开发。国外 Google 公司内部日常运维的网络工程师已经要求必须具备 NetDevOps 的编程能力。

NetDevOps 实际上适合运用在任何存在网络的环境里，包括传统园区网、新型数据中心，以及当下日益火热的 SD-WAN 市场。

1.2.3 为什么我们需要 NetDevOps

为什么我们需要 NetDevOps？原因有以下几方面。

首先，上层业务的自动化程度越来越高。这些自动化的业务推动了网络管理需适应其业务自身的快速迭代与发展。

其次，网络设备及网络技术的演进。最为典型的就是 SDN 的发展让网络设备的 API 越来越丰富，这点拉动了网络管理方式的变革——可以更多地使机器和代码参与到日常的网络管理工作中来。

再次，网络节点数量在急剧增加，导致重复性工作越来越多。只有使用机器和代码才能快速高效地完成此类工作。最后，人工管理网络势必会不可避免地带来大量人为疏忽性错误。网络维护操作中的小失误常常会引起大范围的网络故障，因此网络维护的准确率是非常重要的。就这点而言，使用机器和代码完成自动化将会是更好的选择。

纵观当前的 IT 发展状况：**快速服务是目的，自动化是手段，NetDevOps 是方法。**NetDevOps 是之前游戏规则的改变者，它打破了传统 IT 各领域的边界，其本质是一种思路、一套方法论。如今许多网络和 IT 运维人员已经开始寻求和部署自动化的解决方案，并借此来提升用户体验，提升服务质量和效率。当然这些技能也会提升网络运维人员自身的竞争力。

上士闻道，勤而行之。既然已经有了这么强大、这么好用的技术和理念，我们有什么理由仍然拒之门外呢？

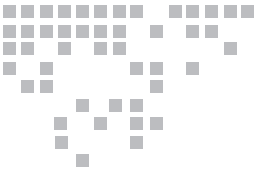
1.2.4 NetDevOps 需要什么样的人

传统的网络工程师需要掌握很多的网络基础知识。例如，OSPF、ISIS、BGP 等路由协议，还有相关设备厂家的命令行使用方式等。既然希望借助 NetDevOps 让机器和代码参与到网络工程师日常的管理与运维工作中，那么必然也需要要掌握一些 NetDevOps 相关的基础知识，如文本处理工具、网络设备的 API、一些简单的编程语言以及一些现成的自动化工具。**本书的目的就是帮助传统网络工程师了解和学习这些方面的基础知识。**即便是传统网络工程师不愿转向软件开发领域，但了解和掌握一些 NetDevOps 的基础知识也是很有益处的。当你和软件开发、软件设计人员进行交流时，当你希望公司开发部人员能帮你们部门开发一些系统和工具时，这些知识将会很有帮助。

1.3　小结

本章一开始谈了很多关于 SDN 和云相关的内容，看似和 NetDevOps 没有太多关系，但正是 SDN 和云的发展让网络 DevOps 变成了可行的且相当迫切的需求。如果不是 SDN 和云这两个技术的双重驱动力，NetDevOps 发展也许还会再晚一些。

NetDevOps 带着 DevOps 的理念与文化，在传统的网络运维领域也一样会掀起出新的浪潮。在第 2 章，我们会和大家聊聊关于如何开始 NetDevOps 的学习。



Chapter 2 第 2 章

如何开始 NetDevOps

基于第 1 章的内容，我们可以看到 NetDevOps 是网络运维的发展趋势。那么，我们如何开始 NetDevOps 呢？我们需要做哪些准备工作呢？本章会从如下几个方面来帮助大家做好准备工作。

- 文档的管理。无论是采用传统的方式运维网络还是采用写程序的方式来维护网络，都会产生大量的文件，有效地管理这些文档和代码以及相应的版本是 NetDevOps 的开始。
- 编程语言的选择。NetDevOps 包含了一定的程序开发工作，如何选择一个适合自己或自己所在公司的编程语言是一件比较重要的事情。目前编程语言多达 5000 多种，我们怎样才能选择更适合当下的语言呢？
- 自动化平台的选择。对大多数中小企业而言，开发一套相对完整的自动化工具是一项工作量很大的工程。特别是在初期，使用现成的工具将会是一个不错的选择。但这些自动化的平台该如何选择呢？
- 我们需要什么样的编程接口。既然要面向网络设备进行编程开发，那么网络设备需要具备什么样的编程接口才会更容易实现 NetDevops 呢？

我们在开始编程之前先关注一下上述的几个问题，这对后续的学习会更有帮助。

2.1 文档内容与版本管理

首先需要声明的是，版本管理并不限于网络工程师们常说的“设备软件版本的管理”。这里提到的版本管理包括网络规划、网络运维涉及的所有文档、代码和设备相关的软件。具体的内容会在后续进行详细的说明。本节从以下几点进行描述：

- ❑ 版本管理的重要性；
- ❑ 网络管理中哪些内容需要版本管理；
- ❑ 如何实施版本管理；
- ❑ 版本管理常用的工具。

2.1.1　版本管理的重要性

大家在进行网络维护的过程中，也许遇到过拿着一份错误的规划表对设备进行了配置，随后网络出现了故障；也许还遇到过相同角色的设备上存在着不一样的策略。例如，一对 SR（Service Router，业务路由器）和 RR（Route Reflector，路由反射器）之间建了 BGPv4 邻居。但是其中一台设备并没有发送 BGP Community（BGP 路由的一种属性），导致流量进行切换时发生了故障。这些问题都与版本管理有着密切的关系。

其实，版本管理在任何管理领域都是一项非常重要的内容。现代社会存在着大量协作，大家在互相沟通合作的时候需要确保信息的统一准确，因此信息的版本管理是非常重要的。即便是由一个人完成的工作，由于时间的关系也需要做好版本的管理工作。只要在过程中出现了信息变化，就需要有版本管理。版本管理存在着如下两个基本要素。

首先，版本管理需要有一个明确的位置来存放这些信息。在考虑了安全性之后，存放信息的位置应该是便于查询的。

其次，存放这些信息的地方需要有一个好的手段来知道它们的版本信息，既要方便知道最新的版本内容，也要方便查询历史的版本内容。图 2-1 所示就是一种非常糟糕的信息保存方法。单纯通过文件名的方式是比较难于管理的，但是这又恰恰是我们日常管理版本最常用的方法。

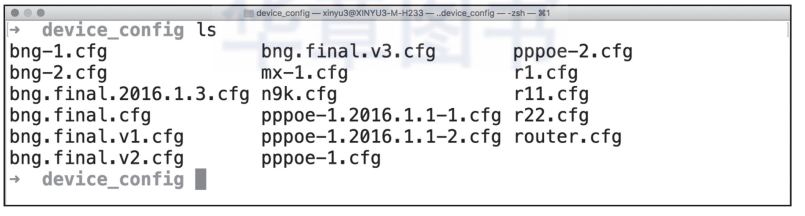


图 2-1　设备配置文件名

2.1.2　需要管理哪些文档

网络管理通常可以分为三个阶段：网络规划、网络建设与网络运维，每个阶段都会产生很多的信息。这些信息很多是通过文档（这里说的文档指一类文件）的形式来体现的。部分公司会采用一些软件系统进行替代，将其保存在数据库中。表 2-1 列出了一些较为常见的文档类型及其常用格式类型。

表 2-1 常见文档

阶段	文档类型	常用格式类型
规划	可行性研究	Word、Excel、PowerPoint
	测试方案	Word
	概要设计	Word、Excel、PowerPoint
	详细设计	Word、Excel
	网络规划拓扑图	PowerPoint、Visio
	资源规划	Excel
建设	资源分配表	Word、Excel
	设备详细配置	TXT
	网络拓扑图	PowerPoint、Visio
	变更方案	Word
	变更脚本	Excel、TXT
	变更操作日志	TXT
运维	应急方案	Word、Excel
	资源使用情况	Excel
	运维事件记录	Excel
	设备配置的备份	TXT

除了上述的文件外，还有一种文件是二进制文件。它们通常由其他公司提供，并不是网络管理者自己产生的。对于这样的文件，通常只需要管理其版本信息即可。

上面描述的这些文档都是需要进行版本管理的。这些文档的类型基本上是 Office 文档的格式或者纯文本的格式。

2.1.3 如何实施版本管理

如果有一些系统能进行版本管理是最好的，但是很多时候并不具备此类完善的系统，这就需要网络工程师自己来管理这些文件和版本信息。管理好这些文件以及它们的版本信息是 NetDevOps 的基础。如何在没有管理系统帮助的情况下快速而有效地进行管理呢？首先，我们需要对这些文件按照格式进行分类。这些文件大体可以分为以下三类：

- 微软 Office 格式的文件，如规划与描述的文件；
- 纯文本文件，如设备配置与一些脚本文件；
- 二进制文件，如网络设备使用的软件操作系统。这一类文件并没有包含在表 2-1 中。这是因为这种文件通常是由设备厂商提供，并不是网络管理者自己产生的。

其次，笔者建议尽量使用纯文本的格式来编写文档。Office 格式的文件通常使用文件名和文件中的版本信息来进行管理。例如，文件名为《 XXX 网络实施总体方案 V1.2》。在文件的开头会有如图 2-2 所示的内容。这样的版本信息是比较常见的版本管理方法。通过阅

读文件中的版本信息，我们可以非常清晰地了解这个文件的相关版本信息。它适合在不同的人员之间、不同的部门，乃至不同的公司之间进行单个文件的传递。虽然我们可以在“文档变更过程”这里找到一些文档修改的简要信息，不过采用这样的方式很难在文件中保留全部的历史细节信息。因此，一些公司会使用微软 SharePoint（https://products.office.com/zh-cn/sharepoint/collaboration）工具来进行团队的文档管理。它可以提供历史版本的管理，以及不同版本之间的比较信息。

版本控制信息

文档属性

属性	内容
项目名称	XXX 公司 2016 年 IP XX 网 XXXX 扩容 XX 期工程
文档标题	XXX 网络实施总体方案
文档版本号	V1.2
版本日期	2017.3.15
作者	张三
审核人	李四

文档变更过程

版本	更新日期	更新人	主要更新内容
V1.0	2017.2.27	张三	初稿
V1.1	2017.3.2	张三	修改内容为 XXXXX
V1.2	2017.3.15	张三/李四	修改内容为 XXXXX
……	……	……	……

图 2-2　文档版本信息

在网络管理中常常会遇到网络拓扑图信息，这样的内容是比较难以管理的。这里笔者推荐大家使用 DOT 文件格式进行网络拓扑的绘制。DOT 是一种描述图形的语言，它能用一种简单的方式来描述图形，而且能兼顾人和机器同时读取和处理。

例如，在代码清单 2-1 中描述一个简单的 IDC 的拓扑：

代码清单2-1　一个简单的拓扑

```
graph G {
    spine1 [label="核心1" color=blue]
    spine2 [label="核心2"]
    leaf1 [label="接入1"]
    leaf2 [label="接入2"]
    leaf3 [label="接入3"]
    leaf4 [label="接入4"]
    leaf5 [label="接入5"]
    leaf6 [label="接入6"]
        spine1 -- leaf1 [color=red];
        spine1 -- leaf2;
        spine1 -- leaf3;
```

```
        spine1 -- leaf4;
        spine1 -- leaf5;
        spine1 -- leaf6;
        spine2 -- leaf1 [color=red];
        spine2 -- leaf2;
        spine2 -- leaf3;
        spine2 -- leaf4;
        spine2 -- leaf5;
        spine2 -- leaf6;
}
```

解析这个 DOT 文件在 Chrome 浏览器（需要安装 DOT Lang Viewer 插件）中的显示结果如图 2-3 所示。使用 DOT 语言可以借助文本的方式来保存一个网络拓扑图。http://www.graphviz.org 提供了 DOT 的详细说明，读者可以参考。

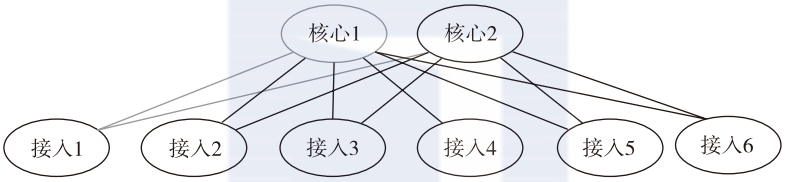


图 2-3 Chrome 浏览器显示拓扑图

再者，对于纯文本的文件，建议读者学习和了解 MarkDown 的文件格式。MarkDown 具有以下优点。

- **兼容性好**。纯文本的文件格式兼容性非常好，任何平台下都可以完全兼容。
- **可以转化为其他格式**。MarkDown 可以很容易地转换为 HTML、PDF 等文件格式，并且经过了一定的排版和格式处理。
- **语法简单**。MarkDown 的语法非常简单，很容易学习和应用。

最后，二进制的文件相对不太好管理。建议大家最好保留每个二进制文件的 MD5 或者 SHA 的 HASH 值。Linux 和 MAC OSX 很容易对此类系统文件进行 MD5 与 SHA 值的计算。例如：

```
$ md5 iosxrv-k9-demo-6.1.2.qcow2.tgz
MD5 (iosxrv-k9-demo-6.1.2.qcow2.tgz) = 71d3be46fb68f8058b6c683f80a0f410
```

通过管理文件的 HASH 值可以确定一个文件的内容是否完整。即使文件名称被修改了，HASH 值也不会发生变化。其值不变就可以认为是相同的文件。大家熟悉的云网盘也是通过这个方法进行海量文件管理的。

2.1.4 版本管理的工具

在版本管理部分，存在很多的工具。目前较为流行的一个工具是 Git（https://git-scm.com）。Windows、Linux 和 MAC OSX 都支持 Git。Git 是一个分布式的版本控制软件，由

大名鼎鼎的 Linux 之父 Linus Torvalds 所开发，并于 2005 年以 GPL 方式发布，其最初目的是更好地管理 Linux 内核开发。现在这个工具几乎是最为流行的版本管理软件。目前 GitHub（https://github.com）是一个通过 Git 进行版本控制的软件源代码托管服务中心。现在 GitHub 不单单有软件的源代码，很多软件的使用说明也通过 GitHub 进行管理。这些内容通常被放在 Git Pages（https://github.io）中。对于本书的读者而言，掌握基本的 Git 工具是后续章节学习的基础。出于篇幅的考虑，这里不对 Git 的使用方法进行展开，读者可以参考 http://rogerdudler.github.io/git-guide/index.zh.html，这个文档是一个非常简洁的 Git 入门教材，其还包括多语言的版本。本书的后续章节中也会遇到 Git 相关命令，本书后续会默认大家已经了解和熟悉 Git 的基本命令。

2.2　编程语言的选择

NetDevOps 中有开发的部分，开发就必然会涉及编程语言。网络工程师在初次接触 NetDevOps 时，都会遇到编程语言的选择问题。下面我们就来简单讨论一下如何选择编程语言。这里我们将从程序语言的选择和数据描述语言的选择两个部分来进行叙述。

2.2.1　程序语言的选择

在程序员的世界里，讨论哪种编程语言是最好的语言，往往会引起非常激烈的争吵。笔者不敢在这里和大家讨论哪种语言是最好的语言，而是从自己的角度和大家分享一下如何选择适合 NetDevOps 的编程语言。

众所周知，编程语言既有编译型语言，也有解释型脚本语言。通常来说，编译型程序执行速度快，同等条件下对系统要求相对较低，C、C++、C# 就是这类语言。相对于编译型语言的存在方式，解释型语言的源代码不是直接翻译成机器语言，而是先翻译成中间代码，再由解释器对中间代码进行解释运行。例如，Python、Ruby、JavaScript、Perl、Shell 等就是解释型语言。

在网络运维和管理中，程序的执行效率也许不是最为关键的。换句话讲，再慢的程序（只要不出错）也会比人工执行效率要高。程序大量的执行过程是需要和网络设备进行交互的。脚本程序在执行过程中很多时候都是在等待网络设备侧返回的数据信息，因此执行效率瓶颈往往并不是语言的代码执行效率。另外，工程师也许更加关心的是程序开发的效率、代码的可读性方面。从这些方面考虑，解释型语言是首选语言。

在众多的解释型语言中，哪些语言更加合适呢？现在程序在开发的时候，通常会采用 Web 方式进行程序发布。基于 Web 的程序通常会分为前端和后端两大部分。前端开发主要解决视觉的问题，即如何更好地展现数据。在浏览器中展现数据，JavaScript 几乎是唯一的选择，当然还有 HTML 以及 CSS 等语言的方式。除了前端，后端的内容也是非常重要的。后端的主要功能是获取、整理以及保存数据。在某些时候，为了简化开发的工作量，只保

留后端的部分也是很正常的。对于 NetDevOps 网络工程师而言，如果只是为了快速地完成工作内容，将需要更多地关注后端的开发。在后端开发中，较常见的语言有 Python、Ruby、Bash、Java。其中，Python 与 Ruby 在 Web 后端开发中非常有优势。另外，NetDevOps 工程师还需要和网络设备打交道，网络设备的厂家很多。近几年，网络设备的可编程能力越来越强。基于笔者的观察，大量的网络设备会支持 Python 与 Bash 两种语言。

因此，笔者建议 NetDevOps 可以选择 Python 与 Bash。如果要兼顾前端的开发，那么也需要了解和掌握一些 Javascript、HTML、CSS 相关知识。在本书的后续章节中，使用的语言主要是 Python 与 Bash，我们会分别介绍一下这两种语言的基本语法，以及在网络运维和管理中基于这两种语言开发的常用一些工具。

2.2.2 数据描述语言的选择

2.2.1 节提到了编程语言的选择问题，其选择余地相对较大，并且笔者给出了认为更加适合的语言，以减少初学者在语言选择上的徘徊。但在数据描述型语言的选择方面会少很多，这里笔者给出一些常见的、在 NetDevOps 开发中通常会用到的格式。这里提到的数据描述型语言，读者最好能较好地掌握（其实掌握这些内容比学一门编程语言要简单很多）。

数据描述型语言主要是为程序提供服务的，也就是说程序能够快速方便地解析它们。部分数据描述型语言还兼顾了人的可读性，让人也能够较为方便地编写和阅读其内容。

下面是常见的数据描述型语言。

1. JSON

JSON（**J**ava**S**cript **O**bject **N**otation）是一种轻量级的数据交换语言，以文字为基础，且易于让人阅读。JSON 数据格式与编程语言无关，它脱胎于 JavaScript，但目前很多编程语言都支持 JSON 格式数据的生成和解析。JSON 用于描述数据结构，其形式如下：

```
{名称：值}
```

- 对象（object）：一个对象以“{”开始，以“}”结束。一个对象包含一组非排序的名称与值对。
- 每个对内的名称与值（collection）：名称和值之间使用“:”隔开。
- 多个对（名称与值对）之间使用“,”分开。
- 值的有序列表（array）：一个或者多个值用“,”分区后，使用“[]”括起来就形成了列表。

```
{["hostname": "Router1", "hostname": "Router2"]}
```

JSON 格式表达的是一种树状的数据类型，非常容易被保存到 NoSQL 数据库中，如 MongoDB。在网络编程中，这种结构较为常见，也易于使用。比如在 Cisco Nexus 系列的交换机上就可以直接输出为 JSON 格式的数据。

```
{
  "ins_api": {
      "type": "cli_show",
      "version": "0.1",
      "sid": "eoc",
      "outputs": {
          "output": {
              "body": {
                  "hostname": "switch"
              },
              "input": "show switchname",
              "msg": "Success",
              "code": "200"
          }
      }
  }
}
```

2. XML

XML（**Ex**tensible **M**arkup **L**anguage，可扩展标记语言）是一种标记语言，用于传送及携带数据信息，不用于表现或展示数据。这样的结构并不太适合人直接阅读。但 XML 中的 tag 是可以携带多个属性的。并且，XML 还有 namespace 等概念。相比 JSON，XML 能表示更加复杂的数据结构，也比 JSON 复杂很多。XML 格式是 NETCONF 协议的默认交换数据的格式。比如 Juniper 的路由器、交换机等网络设备上运行的 JUNOS 可以直接输出 XML 格式的数据。XML 是一种结构化的文本，非常适合程序进行处理。

```
user@host> show chassis alarms
No alarms currently active
user@host> show chassis alarms | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/10.4R1/junos">
  <alarm-information xmlns="http://xml.juniper.net/junos/10.4R1/junos-alarm">
      <alarm-summary>
          <no-active-alarms/>
      </alarm-summary>
  </alarm-information>
  <cli>
      <banner></banner>
  </cli>
</rpc-reply>
```

3. YAML

YAML（http://yaml.org）是一种用来表达数据序列的格式，对人而言可读性较高，可以简单表达清单、散列表、标量等数据形态。它采用空白符号缩进的方式，非常适合用来表达或编辑数据结构、配置文件、文件大纲等内容。由于 YAML 使用空白字符和分行来分隔数据，因此它特别适合用 grep、Python、Perl、Ruby 等语言进行操作。另外，YAML 没有使用各种封闭符号，如引号、各种括号等，这些符号在嵌套结构中会变得复杂并且难以辨

认。YAML 的语法非常简单，主要注意文本的缩进。读者可以参考如下链接：

- http://www.yaml.org/spec/1.2/spec.html；
- http://docs.ansible.com/ansible/YAMLSyntax.html。

下面的 Ansible 的配置文件就采用了 YAML 格式进行编写。

```
---
- hosts: "core"
connection: local
remote_user: "admin"
gather_facts: False
tasks:
- nxos_interface:
  interface: "{{ item }}"
  mode: layer3
  admin_state: up
  transport: nxapi
  host: 10.255.0.75
  username: admin
  password: cisco12345
with_items:
- Ethernet1/1
- Ethernet1/2
- Ethernet1/3
- Ethernet1/4
```

4. YANG

和前面三种语言不同，YANG 不是一种数据描述语言，而是一种数据建模语言。也就是说，它是用来定义数据的数据结构的，而不是数据的实体。通过下面这个例子可以很容易理解。

```
module acme-system {
  namespace "http://acme.example.com/system";
  prefix "acme";
  organization "ACME Inc.";
  contact "joe@acme.example.com";
  description
      "The module for entities implementing the ACME system.";
  revision 2007-11-05 {
      description "Initial revision.";
  }
  container system {
      leaf host-name {
          type string;
          description "Hostname for this system";
      }
      leaf-list domain-search {
          type string;
          description "List of domain names to search";
```

```
        }
        list interface {
            key "name";
            description "List of interfaces in the system";
            leaf name { type string;}
            leaf type { type string;}
            leaf mtu  { type int32; }
        }
    }
}
```

这个文件使用 YANG 定义了一个交换机的输出数据结构。system 包含 host-name、domain-search 以及 interface 三个子内容。下面的输出内容是某一台交换机真实的输出结果。

```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<system xmlns=" http://acme.example.com/system">
  <host-name> Swith-1 </host-name>
<domain-search> abc.com </domain-search>
<domain-search> abc.net </domain-search>
<interface>
  <name>
    <name> Eth3/1 </name>
    <type> Ethernet </type>
    <mtu> 1500 </mtu>
</name>
<name>
   <name> Eth3/2 </name>
   <type> Ethernet </type>
   <mtu> 1500 </mtu>
</name>
</interface>
</system>
</data>
```

通过这个例子，我们可以看出：YANG 文件定义的是一个数据结构，而设备输出的结果使用了这个定义的数据结构，并在结构中相关部分给出了具体的值。关于 YANG 语言具体的内容，读者可以参考 http://www.yang-central.org。

目前，由于各厂家网络设备的输出格式都不一样，并且大量的格式还是非结构化的数据格式，这种非结构化的数据格式对程序的开发并不友好。YANG 语言是专门为网络环境而开发的语言，它对网络设备的输出数据进行抽象化并提供了一个通用的语言。Google、AT&T、Microsoft、BT 等公司共同参与了 OpenConfig（http://openconfig.net）项目，这个开源的项目定义了大量和网络相关的数据结构描述文件。

关于这些数据描述语言，我们在第 9 章中会进行更加详细的说明，并且会给出使用 Python 语言如何处理这四种用于数据描述的格式。

2.3 自动化工具的选择

对于大多数没有丰富编程经验的网络工程师来说，一开始就通过编程的方式来管理网络是一个不小的挑战，也许经过几个月的学习也无法开发出一个能用的小工具，这对初学者来说是一个不小的打击。那么如何能快速地开始 NetDevOps 的相关工作呢？选择一个相对成熟的自动化工具是一个不错的选择。自动化工具有很多，这里笔者将给大家介绍四个常见的工具。

2.3.1 Ansible

Ansible（https://www.ansible.com）是 RedHat 旗下的一个自动化运维工具平台，可以用来做系统配置管理，批量对远程主机执行操作指令。其通过 SSH 协议实现远程节点和管理节点之间的通信。因此，只要是管理员通过 SSH 登录到一台远程主机上能做的操作，Ansible 都可以做到。现在 Ansible 是一个开源的软件项目。其商业版为 Tower，商业版除了可以得到 RedHat 的支持，其还是一个有丰富 GUI 的版本。下面我们以开源的项目为主进行介绍。

Ansible 从 2.1 版本开始就集成了大量网络设备相关的模块。目前 Ansible 的最新版本是 2.3（2007 年推出）。正是这些网络设备模块的集成，使得 Ansible 在网络领域得到了广泛的应用。表 2-2 列出了 Ansible 2.3 支持的网络模块以及这些模块对应的厂商和型号。

表 2-2 Ansible 2.3 网络模块

模块名称	厂家以及型号	模块名称	厂家以及型号
A10	A10 Networks	Eos	Arista Networks
Aos	Apstra Networks	F5	F5
Asa	Cisco Systems ASA	Fortios	Fortinet
avi	AVI Networks	Ios	Cisco Systems IOS Device
Bigswitch	Big Switch Networks	Iosxr	Cisco Systems IOS-XR Device
Citrix	Citrix	Junos	Juniper Networks
Cloudengine	Huawei CE Switch	Nxos	Cisco Systems Nexus
Cumulus	Cumulus Networks	Openswitch	Open Switch
Dellos10	Dell OS 10	Ovs	Open vSwitch
Dellos6	Dell OS 6	Sros	Nokia SR
Dellos9	Dell OS 9	Vyos	Vyos

Ansible 使用的开发语言是 Python，现在 Ansible 已经同时支持 Python 2 与 Python 3。其是通过编写 YAML 文件来定义 Playbook（剧本，即任务列表）的。Ansible 不需要在设备上安装 Agent 软件，但在绝大多数的情况下，还是需要被管理的机器上有 Python 的运行环境和一些基本的 Python 模块。对于网络设备的管理，由于网络设备通常不具备 Python 的运

行环境，因此，在网络设备的模块中，被管理设备可以没有 Python 的运行环境。是否需要 Python 的运行环境，取决于具体的模块。

2.3.2　Puppet

Puppet（https://www.puppet.com）是一个可以用于多平台环境下的集中式系统配置管理平台，能够对基础设施实现自动化的管理。Puppet 通过对整个系统中的各个节点希望达到的最终状态进行描述（检查），然后由 Puppet 执行达到目标。这种思路和过程式程序有着明显的不同。编写过程式程序需要清楚地知道每一步的执行过程并达到最终的目的。举例来说，某台交换机需要增加一个 VLAN 10。基于过程的程序流程如下：登录网络设备；进入设备的配置模式（或者再进入 VLAN 配置模式中）；向设备发送增加 VLAN 的配置；最后在网络设备上提交和保存配置。这是基于过程的一个简单的流程。如果设备已经存在需要添加的 VLAN 10，那么需要在上述的流程中加入更多的内容。对于 Puppet 而言，只需要在其配置文件中增加 VLAN 10 就可以了。Puppet 负责检查交换机上的 VLAN 信息。如果 VLAN 不存在，则会被添加。这样的好处是，让网络的管理者只需要关心最后的终态，而无须处理中间的过程。如何坚持设备的 VLAN 信息，如何添加 VLAN 的配置，是由 Puppet 内的代码来完成的。

Puppet 的开发语言是 Ruby，Puppet 需要在被管理的设备上安装一个 Agent。 Puppet 的服务器和被管理设备之间使用了基于 HTTPS 的 XML RPC 协议通信。目前，已经有很多的网络厂家的设备可以安装 Puppet 的 Agent。读者可以在 https://forge.puppet.com/ 查找自己正在使用的网络设备是否有 Agent 模块。下面给出一个 Puppet 关于增加 VLAN 10 的配置文件。

```
vlan { 'resource title':
  name => 10
  ensure => present
  description => VLAN10
  provider => Cisco
}
```

2.3.3　Chef

Chef（https://www.chef.io）是 Ruby 与 Erlang 开发写成的配置管理软件。Chef 相当于一个脚本管理工具，但功能要强大得多，可定制性强。 Chef 将脚本命令代码化，定制时只需要修改代码，其安装过程就是执行代码的过程。

Chef 主要包括三大块：Workstation、Chef Server、Chef Client（Node）。

Workstation 通常是使用者的工作电脑，使用者在 Workstation 中创建 chef-repo，并且上传到 Chef Server，chef-repo 包括 cookbooks、recipes、roles、environment 等内容。cookbooks、recipes、roles 是 Chef 对基础设施的抽象化定义。

Chef Server 用来存储 Workstation 上传的各种资源，包括 cookbooks、roles、environments、

nodes 等。我们可以使用公有的 Server，也可以通过开源项目搭建自己的企业服务器。Chef Server 提供了非常丰富的 API，用于与 Workstation 和 nodes 传输资源和数据。 Chef Server 原本由 Ruby 来实现，但后来为了保持高并发和稳定性，以及能够同时服务更多数量级的 nodes，Chef Server 内核改用了支持高并发的 Erlang 程序。

Chef Client 是安装在 Node 上的一个软件。Node 是基础设施中的一台服务器，即 Chef 管理的机器。一个 Node 可以是一台物理服务器、一台虚拟机，甚至是一台交换机或路由器。如果你想要在 Node 上部署环境，那么 Node 会与 Chef Server 进行交互以获取信息，并在 Node 上执行环境初始化操作。

2.3.4 SaltStack

SaltStack（https://saltstack.com）也是一个配置管理系统，能够维护预定义状态的远程节点。SaltStack 的核心功能如下：

- 使命令发送到远程系统是并行的而不是串行的；
- 使用安全加密的协议；
- 使用最小、最快的网络载荷；
- 提供简单的编程接口。

SaltStack 执行程序可以为纯 Python 模块。SaltStack 执行过程中收集到的数据可以发送回 master 服务端，也可以发送到任何程序。SaltStack 可以被一个简单的 Python API 调用，或者从命令行直接调用，所以 SaltStack 可以用来执行一次性命令，也可以作为一个更大的应用程序的一个组成部分。

从结构上看，SaltStack 与 Ansible 无 Agent 的设计相反，SaltStack 在部署上可以分为 master 和 minion 两个部分，其中 master 相当于统领所有机器的总管，而 minion 则是部署在被管理机器上面的 Agent 进程。

2.3.5 如何选择

前面介绍了四个比较主流的自动化工具。这些工具一开始都是针对服务器而开发的，对网络设备支持的能力有限。但是，随着 NetDevOps 需求日益增长，这些自动化工具开始支持网络设备。具体选择哪一个工具可以从以下几个方面来考虑。

- 所在公司使用了哪个自动化工具。尽可能地使用公司已经成熟的自动化工具。
- 使用者熟悉哪个工具。也许你在维护网络的同时，还需要维护很多的服务器，那么用原来已熟悉的工具更好。
- 使用者或开发人员更加熟悉 Ruby 还是 Python 语言。在使用了一段时间后，也许会提出更多的需求，那么熟悉工具的开发语言对后期的二次开发会有一定的帮助。当然这也不是绝对的。例如，Chef 中使用了大量的 RESTful API，理论上可以用任何语言做二次开发。

❑ 网络设备支持的情况。这需要和设备厂家一起进行评估。

如果读者对上述的几点完全不在意的话，那么笔者推荐大家使用 Ansible 来进行网络设备的管理。理由有以下几点。

首先，Ansible 提供开源的版本，也提供商业版本。使用者可以根据自己的需求进行选择。商业版本可以获得更多的技术支持。

其次，Ansible 使用 Python 语言进行开发，配置文件使用 YAML 的文件格式。为什么笔者在 NetDevOps 中更加倾向于 Python 开发，这点可以参考 2.2 节。

再次，Ansible 使用 SSH 作为通信协议。这个协议是目前网络设备大量使用的协议。使用这个协议，不用更改现在对网络设备的管理方式。并且，Ansible 支持的网络厂家的设备类型也越来越多。而且，Ansible 可以很容易地兼容 NETCONF 的协议，因为大部分厂家的 NETCONF 协议都是基于 SSH 进行通信的。

最后，Ansible 不需要在网络设备上安装任何的 Agent。这也许是最为重要的一点。

2.4　网络设备的编程接口

前面我们讨论了开始 NetDevOps 之前的一些知识点，这些内容基本上还没有涉及具体的网络设备这一侧。既然是 NetDevOps，我们必然需要和网络设备打交道。网络设备需要什么能力才能更加适合 NetDevOps？我们如何看待与评估？这是本节将要阐述的内容。

2.4.1　网络设备接口的分类

网络设备的核心任务是转发数据包，网络工程师的核心任务是控制网络设备按照预定的设计转发数据包。网络工程师在控制网络设备的时候有以下三类方式：

❑ 传统的接口；

❑ 路由协议层面；

❑ 数据包层面。

（1）传统的接口类型

对于网络工程师而言，最传统的方式是通过命令行的方式修改设备的配置。修改设备配置的接口有多种，最常见的是用 Telnet、SSH 或者直接用 Console 口登录到设备上进行操作。除此之外，许多厂家的设备可以通过 SNMP 的 write（写）能力对设备进行配置修改。对于支持 NETCONF 的设备，也可以通过 NETCONF 接口发送 XML 或 JSON 格式的内容对设备进行配置修改，从而达到改变设备转发路径的目的。同理，部分厂家使用 NETCONF 的方式也可以归结到这一类接口。这一类接口主要用于处理非结构化或结构化的文本信息，在后续的章节中，我们会详细讨论如何处理非结构化与结构化的文本信息。

（2）通过路由协议或类路由协议的类型

这类方式发送广义的 NLRI（**N**etwork **L**ayer **R**eachability **I**nformation）给设备，设

备在收到这些信息后将其转化为硬件能识别的转发表项下推送到硬件上，从而完成网络设备转发路径的变更。这一类中，较为常见的还有 PCEP（Path Computation Element Communication Protocol）、BGP、BGP-LU 等协议方式。另外，著名的 OpenFlow 协议也属于这一类。OpenFlow 发送给网络设备的流表（flow table）并不能算真正意义上的转发表，它和 BGP-FlowSpec 在很大程度上还是比较相似的。不过，OpenFlow 拥有更多的字段，能完成更多的事情。例如，OpenFlow 定义的字段存在一些纯控制平面使用的内容，Cookie 就是一个用于控制的标示。这一类接口主要处理的是这些广义 NLRI 信息的包结构，以及转发路径和这些包的对应关系。这部分的内容比第一类的接口类型相对复杂一些，本书主要关注的是 NetDevOps 入门的内容，因此不会涉及这一部分的开发。读者如有兴趣可以参考 ExaBGP、RYU 等开源项目。

（3）数据包层面的类型

第三类和前两类不一样，前两类可以归纳为：通过改变设备的转发表而影响设备的转发路径。转发表是可以通过静态配置或者路由协议计算来生成的。而第三类并**不修改大部分网络设备的转发表信息，而是在数据包中添加更多的包头信息，从而实现转发路径的更改**。这类方式最为典型的是 Segment Routing 和 NSH（Network Services Headers）。Segment Routing 通过 MPLS 标签堆栈来增加数据包中的转发信息，网络设备通过读取数据包头中的这些信息获取相应的转发路径。而相比 Segment Routing，NSH（参考 https://tools.ietf.org/html/draft-ietf-sfc-nsh-12）携带了更加灵活的信息。由于它比 MPLS Label 更加地灵活，目前用 ASIC 来实现 NSH 的识别和转发有一定的难度。但是，NSH 非常适合在软件方式的网络设备（如 vRouter、vSwitch 等 NFV 设备）上来实现和完成。这里举两个例子来区分一下第三类和前两类的区别。

【例 1】 运送包裹，见图 2-4。包裹在经过每一个中转站的时候是完全无法控制的，也不知道下一个中转站的情况。每一个中转站通过查询包裹的目的地址，从而知道如何往哪里投递（转发）它。这个过程和现在的 IP 网络非常类似，它和 IP 网一个比较大的区别是物流网络往往有较大的缓存，这些缓存就是物流的中转站。即使中转站爆仓，包裹被丢弃的概率还是非常低的。

周日	09:09:47	卖家发货
	17:44:18	南京六合大厂揽投部 收寄
	19:38:20	南京国内转运中心开拆,业务员：186
	21:56:00	南京国内转运中心封发,发往上海处理中心,业务员:宁速特机
周一	01:45:58	上海处理中心开拆,业务员：091913
	05:21:08	上海处理中心 封发,发往莘庄(本部）,业务员:文件40
	07:50:31	莘庄(本部）出班
	10:03:40	莘庄(本部） 妥投,业务员:

图 2-4 包裹运送例子

【例 2】 汽车导航，见图 2-5。这个例子和例 1 看似有点类似，但是还是有本质的区别。在这个例子中，车行驶的路径在出发前就已经规划好了（假设在出发后，司机并不改变行驶路径）。在这个规划中，每个路口都有一个提示。司机在到达这个路口时会根据这个提示进行操作，完成后这个提示就会被丢弃掉，司机需要关注的只是下一个提示。这个过程和目前的 IP 转发是完全不一样的。Segment Routing 和 NSH 的方式就与汽车导航的方式非常相似。以 Segment Routing 为例，数据包在进入 MPLS SR 域的边缘节点时，数据包头会被加上了一串 MPLS 的标签。这些标签就像图 2-5 中每一个路口的提示信息。

图 2-5　汽车导航

本书不会涉及这部分的应用开发，目前此类的应用相对还比较少，这类的开发也相对复杂，已经超出了 NetDevOps 入门的阶段。但在本书的第 14 章会有一个关于路径计算的案例。在此案例中，当完成路径计算后，会通过 BGP 协议给网络设备发送路由信息，达到控制路径转发的目的。

2.4.2　网络设备编程接口的特征

2.4.1 节中提到了网络设备接口分类的问题，那么网络编程对设备的接口有什么要求呢？这里归纳了编程接口的四个基本特征：

- 结构化数据；
- 无连接与无状态性；
- 事务性；
- 幂等性。

1. 结构化数据

目前网络设备输出的数据主要是面向人的显示方式（大多为 CLI 输出结果），这样格式的数据缺少一些结构化的标识。对于人来说，数据的显示和信息的分类主要是通过换行和空格符进行标识的，但这样的显示方式对于程序而言并不是很方便。程序更加容易处理使用封闭符号（各种括号与引号等）的信息嵌套格式，程序对空格与换行完全没有任何要求。对于使用换行符和空格符格式化的数据，本书暂且称它们为半结构化数据（这里指的半结构化是相对 JSON/XML 而言的）。对于结构化数据以及半结构化数据的处理方法，在本书的 Python 部分会提供详细的介绍。

2. 无连接与无状态性

无连接指的是每次连接后只处理一个请求。服务端处理完客户端的请求，并收到客户的确认后，立即断开连接。无状态指的是对于事务的处理没有记忆性，服务端也不知道客户端是什么状态，服务端只是根据请求发送数据。现在网络设备的 CLI 接口几乎都不具备这样的能力，CLI 方式是强交互式的方式。举例来说，如果我们需要在一台 Cisco 传统路由器上配置一个接口的 IP 地址，首先需要登录到设备上，然后输入命令 enable 进行入 enable 模式，接着输入命令 config terminal 进入配置模式，再输入 interface xxx 进入接口配置模式，到这里才可以进行接口 IP 地址的配置。对设备的这几个请求必须是按顺序的，并且每次请求后都不能和设备断开连接，一旦断开了连接，必须重新开始。

NETCONF 或者 RESTful 的接口基本上实现了无连接与无状态性，如果网络设备支持的话，对于开发而言将会更加方便。

对于不具备无连接与无状态性的网络设备的开发，本书后面的章节会有讨论。对于具备无连接与无状态性的网络设备，本书的后续章节也将会有涉及。

3. 事务性

事务性是指在执行一个操作时，要么成功执行完成，要么根本不执行。假定现在对一台交换机有如下一个事务需要操作：在交换机的上行接口中添加一个 VLAN。对于网络工程师而言，这个事务通常可以拆解为两个小事件。首先在交换机上添加一个 VLAN ID（假定原来不存在这个 VLAN），然后在上行接口中增加这个 VLAN ID。如果这两个小事件不能组成一个事务，就有可能出现如下这样的问题：提交的上行接口名称错误，导致在上行接口上增加 VLAN ID 这个事件失败；但是第一个在交换机上创建 VLAN 这个事件先执行完成，然后又没有事务性的回滚机制，最终导致在设备上留下了这样一个多余的 VLAN ID。此类的操作或许是网络设备上出现很多“垃圾”配置的原因之一。时间久了，大量的、无效的“垃圾”配置将给运维工作带来非常大的痛苦。NETCONF（https://tools.ietf.org/html/rfc6241）中定义的 candidate 配置、commit 以及 Rollback-on-Error 的功能，将能很好地保证设备具备事务性的处理能力。

4. 幂等性（idempotence）

这个概念来自于数学，现在计算机科学也借用了这个概念，其含义是指重复使用同样的参数调用同一方法时总能获得同样的结果。举例来说，下面 permit ip any 2.2.2.2/32 这行命令就不是一个幂等的方法。设备在这个命令前自动添加了一个序列号 20。当 t1 这个 access-list 被其他人修改后，再运行一次 permit ip any 2.2.2.2/32 就会得到完全不一样的效果。

```
switch(config-acl)# show ip access-list t1
IP access list t1
    10 permit ip any 1.1.1.1/32
switch(config-acl)# permit ip any 2.2.2.2/32
```

```
switch(config-acl)# show ip access-list t1
IP access list t1
    10 permit ip any 1.1.1.1/32
    20 permit ip any 2.2.2.2/32
```

这里修改了 t1 这个 access-list 后，重新运行了一次 permit ip any 2.2.2.2/32，就得到了完全不一样的结果。这条命令如果修改为 20 permit ip any 2.2.2.2/32，就会解决这个问题，不过要避免序列号 20 这个在 acl 中没有被使用，那么每次运行这条命令就可以得到完全一样的结果。

```
switch(config-acl)# show ip access-list t1
IP access list t1
    10 permit ip any 1.1.1.1/32
    30 permit ip any 3.3.3.3/32
switch(config-acl)# permit ip any 2.2.2.2/32
switch(config-acl)# show ip access-list t1
IP access list t1
    10 permit ip any 1.1.1.1/32
    30 permit ip any 3.3.3.3/32
    40 permit ip any 2.2.2.2/32
```

这个例子就是幂等性的问题，这在编程时需要注意。

对于上述四个特性，如果网络设备的 API 都能直接支持将是最好的。但是在很多情况下，往往不尽如人意，很多时候我们不得不通过传统的 CLI 方式与网络设备进行交互。当然，即使没有上述的特性，也不代表我们不能通过程序来实现一些功能，只不过我们在编写程序的时候需要处理更多的内容。

2.5　小结

本章主要和大家介绍了在开始 NetDevOps 之前应该了解的一些准备工作。首先，管理好自己的文档，良好的文档管理是 DevOps 的前提。其次，本书介绍了编程语言选择的问题。另外，如果读者还不具备编写程序的能力，或者所在的网络规模比较小，可以尝试先用一些开源的自动化工具来实现一些小功能，代替那些经常需要重复的工作。本书的第二篇还会介绍一些 Linux 下的常用工具，希望这些工具也能帮助大家提高工作效率。最后，本书讨论了关于网络设备编程接口的情况，希望这部分的内容能帮助大家选择合适的编程接口。

从第 3 章开始，我们将通过大量的例子和大家一起开始 NetDevOps 之旅。

HZ BOOKS
华章图书

第二篇 Part 2

基 础 篇

基于 NetDevOps 进行网络管理，我们需要搭建合适的工作环境和合适的开发环境，这是后续开展工作和学习的基础。因此，本篇会从如下几章来介绍如何构建我们的工作环境和使用一些常用工具。

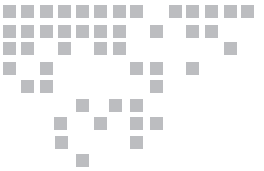
第 3 章，介绍如何在 Linux 环境下开展日常工作，如何在 Linux 管理登录设备的会话，如何使用 Telnet 工具，如何使用 SSH 工具连接设备和建立一些 SSH 隧道。

第 4 章，介绍一些 Linux 下的一些常用工具。这些工具可以帮助大家获取网络设备的一些信息，例如，通过 SNMP 协议获取网络设备的信息，获取网络的可达性信息，获取网络的 traceroute 信息。

第 5 章，介绍使用 Linux 下的常用工具来处理网络设备输出的文本内容。这里使用的工具是 Linux 文本处理的三大利器：grep、awk 和 sed。本章最后还对 vi/vim 进行了简单介绍。

第 6 章，介绍使用 Docker 来搭建 TFTP、DNS 以及 DHCP 服务器。这些服务都是网络工程师常用的一些服务。我们可以利用 Docker 来构建自行开发的工具。

希望本篇的内容能够帮助广大的网络工程在 Linux 环境下进行日常运维和开发工作。



Chapter 3 第 3 章

认识命令行工具

命令行也叫 CLI（**C**ommand **L**ine **I**nterface），是网络工程师最为熟悉的部分。网络工程师在管理网络设备时，首先需要和设备之间建立通信会话。对于网络设备而言，网络工程师使用较为广泛的有串口通信协议（常常称为 Console）、Telnet 和 SSH。随着网络规模的增加，网络管理者需要管理的设备数量也在不断增加，如何更好地管理这些会话已变得越来越重要。

在当前网络环境中，通常需要在安全性和便利性中间找到一个平衡点。当我们在管理网络设备时，通常需要先登录到一台堡垒机，然后通过这台堡垒机登录到具体的网络设备。这样的堡垒机通常是 Linux 平台的服务器，它除了可以提高设备管理的安全性，还可以让我们在服务器上部署一些网管软件、管理工具，甚至是一些可自动化执行的脚本。

对于有一定规模的网络，图 3-1 给出了其网络管理的抽象结构。网络管理人员不能直接登录到生产网的网络设备，而是必须通过堡垒机登录到网络管理服务器或 Console 服务器（Console/Terminal Server），然后登录到具体的网络设备。在图 3-1 中，网络管理网包含了版本管理服务器（图 3-1 中为 Git Server，其他版本管理软件也可以）。版本管理服务器上至少保存三个方面的内容。

1）所有设备定期配置备份和每次变更后的配置。

2）所有设备的操作日志。

3）所有变更过程使用的脚本文件。脚本文件可以由设备配置文件组成，也可以由代码组成或者是自动化工具的编排文件。

这样设计的原因是基于以下几个方面的考虑。

- 让所有的操作都能留下痕迹；

- 所有的设备配置、代码和脚本集中存放将有利于团队协作；
- 集中提供一个代码运行的环境；
- 为后续功能扩展提供逻辑空间。

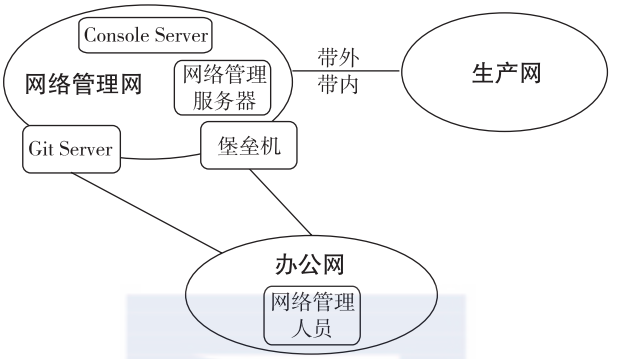


图 3-1　网络管理的抽象结构

对于小规模的网络，图 3-2 给出了更加简单的抽象结构。我们可以看到，最简化的抽象结构包含了堡垒机和版本管理服务器（图 3-2 中为 Git Server）。另外，Console 服务器则是尽可能提供，通过 Console 端口接触设备的方式是很重要的，尤其是在一些网络故障的情况下。如果在网络中实现了相对完善的 ZTP（Zero Touch Provisioning，零接触部署）和 ZTR（Zero Touch Replacement，零接触替换）环境，Console 服务器可以被省略。对于 ZTP/ZTR，本书不会涉及其细节部分，相信读者完成本书的全部内容后可以自行开发出适合自己日常应用的 ZTP/ZTR 系统。

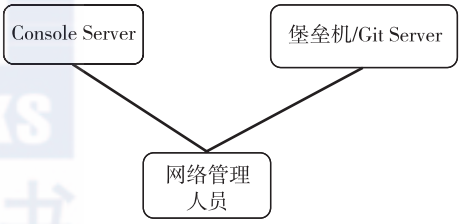


图 3-2　简化网络管理的抽象结构

下面我们来看看如何通过 CLI 管理网络设备。

3.1　用 screen 实现终端的会话管理

screen 是 GNU 计划开发的用于命令行终端管理的自由软件，其官方网址为 https://www.gnu.org/software/screen/。screen 软件有三个主要功能。

- 会话恢复。只要 screen 进程没有终止，其管理的会话都可以恢复。这在远程登录且网络质量不是很好的情况下非常有用，即使网络发生临时中断也可以恢复到中断前的状态。这和远程桌面（Windows RDP）是类似的，只不过这是一个文字文本界面而非图形界面。
- 支持多窗口。用户可以使用 screen 软件在一个会话下同时连接到多个远程虚拟终端。如果还拿远程桌面做比喻，多窗口就像一个远程桌面里面有多个应用程序的

窗口。

- 会话共享。screen 可以允许多个用户同时登录到一个会话中，在这个会话中可以看到完全一样的输出，也可以多方同时输入。当然，这个窗口的方式是可以提供权限控制的。这样的功能在多人共同处理一个问题时会带来很大便利。

会话（session）与窗口（window）的概念

一个会话就是一个 screen 的进程。一个会话可以有多个窗口，每个窗口是一个伪终端。在本节中会话与窗口就是指上述的两个含义，但在其他章节也许会有其他的含义。

3.1.1 安装 screen

screen 并不是所有 Linux 发行版本默认会安装的软件，有时候用户需要自己安装。这里给出 CentOS 与 Ubuntu 的安装方法。

1）CentOS 的安装方法：

```
[root@Centos7 ~]# yum -y install screen
//如果你不是root用户使用如下命令
[yuxin@Centos7 ~]$ sudo yum -y install screen
//通过下面这个命令查询是否安装成功
[root@Centos7 ~]# rpm -qa screen
screen-4.1.0-0.23.20120314git3c2946.el7_2.x86_64
```

2）Ubuntu 的安装方法：

```
yuxin@Ubuntu:~$ sudo apt-get -y install screen
//通过下面这个命令查询是否安装成功
yuxin@Ubuntu:~$ apt list --installed screen
Listing... Done
screen/trusty,now 4.1.0~20120320gitdb59704-9 amd64 [installed]
```

3.1.2 screen 基本语法

这里列出了 screen 命令常用的参数。更加详细的应用可以参考 screen 官方手册。

screen 语法：

```
screen [-Rvx -ls][-d <会话名称>][-h <行数>][-r <会话名称>][-S <会话名称>]
```

说明如下。

- -R：先试图恢复离线的作业。若找不到离线的作业，即建立新的 screen 作业。
- -v：显示版本信息。

  -x ：登录到一个 screen 会话，即使这个会话已经被人使用了也可以同时登录。如果系统不只有一个会话，需要指定会话名称。在多用户环境下需要小心，可能会导致死循环。
- -ls 或 -list：显示目前所有的 screen 作业。

- -d < 会话名称 >：将指定的 screen 会话离线。
- -h < 行数 >：指定缓冲区的行数（和内存大小有关）。
- -r < 会话名称 > ：恢复离线的 screen 会话。如果 screen 会话是一个激活的会话，将无法再次打开。
- -S < 会话名称 >：指定 screen 会话的名称。
- -wipe[⊖]：检查目前所有的 screen 会话，并删除已经无法使用的 screen 会话，用于清理一些死进程。

3.1.3　screen 基本操作

1. 创建会话

安装完后，直接输入 screen 命令就可以启动 screen 会话。使用这样的方式启动的 screen 会话是没有名字的，不方便后续的辨认。因此，通常启动 screen 会话的方式如下：

```
[root@Centos7 ~]# screen -S yuxin
```

启动 screen 会话后，默认会创建第一个窗口，这个窗口的编号是 0（在计算机的世界里，大部分都是从 0 开始编号的），并且它会启动一个系统默认的 shell。如果你对 screen 没有做过任何的配置，那么这个 shell 和你之前启动的 shell 没有什么区别。这似乎让你感觉什么也没有发生，仿佛只是做了一个清屏的动作（clear 命令）。其实，你已经开启了一个 screen。在 screen 命令后可以直接加上你想执行的命令。例如，你想直接登录一台设备（见下面的命令）。当你退出登录这台设备的同时，这个 screen 的这个窗口也就结束了。如果这个窗口恰巧又是这个 screen 的最后一个窗口，那么整个会话也就结束了。

```
[root@Centos7 ~]# screen -S yuxin ssh admin@10.74.82.252
```

2. screen 会话中的常用命令

在 screen 会话中，所有的命令都是以 C-a（快捷键 Ctrl+A）开始的。下面列出的命令形式表示为：先按组合键 Ctrl+A，松开后再按空格键。表 3-1 给出了 screen 常用的命令。这些命令都是系统默认的，我们也可以根据自己的需要进行定制和修改。如何定制和修改这些命令请参考官方的文档。

表 3-1　screen 常用命令

命　令	解　　释
C-a ?	显示所有键的帮助信息
C-a c	创建一个新的窗口
C-a n	切换到下一个窗口

⊖ 并不常用，但可能会用到。没有出现在常用参数中。

（续）

命 令	解 释
C-a p	切换到前一个窗口
C-a 0..9	后面是任意一个数字，直接切换到编号对应的窗口。如果窗口 id 超过了 9，需要在 C-a 后输入一个单引号（即’），然后在提示下输入窗口 id 后回车。或者输入一个双引号（即”），这时会出现一个菜单以供选择。当然这两个方法在窗口 id 没有超过 9 的时候也可以使用
Ctrl+a [Space]	窗口从左到右循序切换
C-a C-a	在两个最近使用的窗口间切换
C-a x	锁住当前的窗口，需用用户密码解锁
C-a d	分离（detach）操作，暂时离开当前会话，将当前的 screen 会话（也许包含了多个窗口）放在后台执行，此时在 screen 会话中，每个窗口内运行的进程（无论是前台 / 后台）都在继续执行，即使退出了当前的服务器也不影响这些进程的运行。但是，如果在一个窗口中登录到了一台路由器或者是交换机，并且这台设备启用了 idle-timeout 的功能，那么这个窗口是会被终止的。如果会话就只有这么一个窗口，那么这个会话将是一个死进程
C-a z	当前的会话放到后台执行，在 shell 中执行 fg 命令则可回到 screen
C-a w	只是显示所有的窗口列表，但无法进行选择。如果需要选择使用双引号
C-a t	显示当前的时间及系统的负载
C-a k	强行关闭当前的窗口。screen 会问你是否真的要 kill 这个窗口，y 表示 yes，n 表示 no

3.1.4 定制你的 screen

screen 提供了丰富且强大的定制功能，其默认两级配置文件的位置是 /etc/screenrc 与 $HOME/.screenrc（ /etc/screenrc 是整台服务器全局的配置，$HOME/.screenrc 是当前用户的配置；当配置冲突时，优先使用 $HOME/.screenrc 里面的配置）。设置 screen 选项、定制命令键、设置会话启动的窗口信息与日志信息、启动多用户模式、设置用户的访问权限等，都可以在配置文件中设置。

```
[root@centos7-1 ~]# cat .screenrc
hardstatus alwayslastline
hardstatus string "%{.bW}%-w%{.gY}%n %t%{-}%+w  %=%{..G} %c:%s %M-%d-%Y"
startup_message off
vbell off
defutf8 on
```

图 3-3 是笔者常用的 screen 配置界面，screen 窗口下方始终显示窗口的名称和系统时间。这项设置是通过上面配置命令中 string 后面的参数来设定的。如果读者有兴趣定制自己的风格，可以参考 screen 的文档，或者在互联网上搜索其他人的配置样例。

3.1.5 用 screen 连接串口

网络工程师经常需要使用 console 串口连接网络设备。Apple MAC OSX 和 Linux 的

系统默认没有类似 Windows 平台下的超级终端软件，当然大家可以选择商业软件（如 SecureCRT），其实也可以使用 screen 软件来连接串口（Console 口）。

图 3-3　screen 界面

下面以 Apple MacBook Pro 计算机为例进行介绍。

首先，需要安装 USB 转串口的驱动。这个需要根据大家购买的串口转接头具体型号而定，只要厂家能提供 MAC 下的驱动就可以。驱动的安装需要根据厂家的指导来完成。

其次，需要把 USB 转串口转接头部件连接到计算机上，然后输入下面的命令用于查找刚刚连接到计算机的设备信息。这个命令用于查询 MAC OSX 计算机包含了哪些串行口的设备。

```
$ ls /dev/cu.*
/dev/cu.irxon-DevB              /dev/cu.usbserial-FT9SI3M5
```

这里 /dev/cu.usbserial-FT9SI3M5 就是笔者的 USB 转串口设备。下面使用 screen 来打开这个串口，命令的最后一个参数是串口需要使用的波特率值。这样就可以连接到 console 设备了，命令如下：

```
$ screen /dev/cu.usbserial-FT9SI3M5 9600
```

在 Linux 环境下，/dev/ttyS0 和 /dev/ttyS1 是主板自带的串口，USB 的串口默认为 /dev/ttyUSB0 或 /dev/tty/USB1 等。在找到这些设备名后，screen 的使用方式与前面 MAC OS 的完全一致。如果读者有兴趣可以使用树莓派（https://www.raspberrypi.org）加多个 USB 转串口的部件来实现一个自制的简易 Console Server[⊖]。最新的树莓派 3 集成了 Wi-Fi 和蓝牙接口，通过它 DIY（Do It Yourself）一个无线的 Console Server 也不是难事。

⊖　具体内容可以参考 https://www.raspberrypi.org/forums/viewtopic.php?f=36&t=50735。

3.1.6 管理 screen 的日志

screen 日志的记录方法有以下几种。

第一种方法：在启动 screen 的时候加上 -L 的参数。

```
[yuxin@Centos7 ~]$ screen -L -S yuxin
[yuxin@centos7-1 ~]$ ls
screenlog.0
```

第二种方法：启动 screen 的时候不加 -L 的参数。启动后使用命令 C-a H，同样会在当前目录下生成 screenlog.0 的日志文件。第一次使用 C-a H 命令，开始记录日志，屏幕左下角会显示 Creating logfile “ screenlog.0”。当第二次使用 C-a H 命令后，停止记录日志，屏幕左下角会显示 Logfile “screenlog.0” closed。

这两种方式都有缺点，其文件名是固定的，不能根据会话名和时间自动创建日志文件名，这为后续的整理带来了许多不便。

第三种方法：修改配置文件。配置文件可以是 /etc/screenrc 或者是 $HOME/.screenrc。我们在配置文件中加入如下配置：

```
logfile $HOME/log/screenlog_%S_%t_%Y_%m_%d_%c.%n.log
```

首先要确保 $HOME 目录下有 log 文件夹。如果没有，可以通过 linux 命令——mkdir $HOME/log 来创建。然后使用如下命令启动一个 screen。

```
[yuxin@centos7-1 ~]$ screen -L -S yuxin -t router1
[yuxin@Centos7 ~]$ ls log
screenlog_yuxin_router1_2017_05_09_14:55.0.log
```

这时我们可以看到在 log 文件夹中产生了一个 log 文件。上述配置文件中各参数含义如下：

- %S 表示会话的名称；
- %t 表示窗口的名称；
- %Y 表示年；
- %m 表示月；
- %d 表示日；
- %c 表示时间；
- %n 表示窗口 id。

3.1.7 多人共享一个会话

在多人协作的情况下，这是一个非常好用的功能。首先，需要开启 screen 的多用户模式。在 /etc/screenrc 配置文件中加入如下配置：

```
multiuser on
```

然后用户 A 启动一个 screen 会话，此时如果允许另外一个用户能够访问自己的会话，需要在这个会话中添加权限，方法是在 C-a 后输入 :acladd <username>。然后使用命令：

```
[yuxin@centos7-1 ~]$ screen -ls
There is a screen on:
    24631.yuxin     (Multi, attached)
1 Socket in /var/run/screen/S-yuxin.
```

另外一个用户如果需要同时登录这个会话，输入的命令如下：

```
$screen -x <username>/24631.yuxin
```

这样，两个用户可以同时看到相同的操作，无论是登录设备的方式是基于串口协议（也称为 console）还是 Telnet 或 SSH。这个功能在多人进行协作时还是很方便的。

screen 是一个会话管理软件，可以很好地管理终端会话。虽然 screen 这个软件并不大，但是其功能非常丰富。上面描述的功能只是一些常用的功能，更多的功能可以参考其手册。除了 screen，tmux（https://tmux.github.io）也是具有类似功能的软件。

3.2　用 Telnet 和 SSH 管理设备

Telnet 与 SSH 是登录网络设备常用的工具，本节将和大家一起回顾这两个工具的使用方法。众所周知，SSH（Secure Shell）是一个应用层和传输层上的安全协议，并包括了一些扩展功能，因此，本节的重点将是 SSH。

3.2.1　Telnet

Telnet 可以算是一个古老的协议，IETF 在 1983 年通过 RFC 854 发布了这个协议。RFC 854 中指出 Telnet 只能工作在 TCP 协议上，UDP 是不支持的。因为 Telnet 采用明文传送数据包，安全性不高，因此笔者强烈建议在管理网络设备时采用 SSH 方式。

Telnet 的使用非常简单，相信广大读者并不陌生，其基本命令如下：

```
$ telnet <host> <port>
```

<host> 可以是 IP 地址或者是主机名，<port> 为 TCP 的端口号。登录设备后，后续根据设备提示输入相关信息就可以了。Telnet 的命令虽然很简单，但是它也有很多参数。下面列出几个常用的参数[⊖]：

- -4：强制使用 IPv4；
- -6：强制使用 IPv6；
- -l< 用户名称 >：指定要登入远端主机的用户名称；
- -S< 服务类型 >：设置 Telnet 连线所需的 IP TOS 信息。

当 Telnet 后面为 IP 地址时，-4 -6 会没有什么意义，当 Telnet 后面为主机名时，-4 -6

⊖ Linux、MAC 和 Windows 平台的 Telnet 命令参数可能会有一些小区别。这里主要是针对 Linux 平台。

才会有意义。很多系统默认会用当前的用户名作为远程登录的用户名，那么用 -l 的参数指定用户名是很有必要的；否则，第一次登录只能选择失败（某些网络设备并不会）。最后，-S 在某些特定情况下可以使用，例如，做 TOS 的功能性验证时，又或者你确实需要使用不同的 IP TOS 值来登录设备，达到更好的管理效果。

除了使用 Telnet 进行设备访问外，Telnet 还可以探测设备或服务器是否开放了某个 TCP 的端口。使用的方式是，直接 Telnet 设备或服务器的 IP 和 TCP 端口。如果在输入命令后，没有得到端口无法打开的信息，那么我们可以初步判断刚刚 Telnet 的端口是关闭的。在 4.2.1 节中，我们还会介绍其他功能更加丰富的工具用于端口的探测。

3.2.2 SSH 介绍

最初的 SSH 协议是由芬兰人塔图·于勒宁在 1995 年设计开发的。SSH 协议框架通过 RFC 4251 发布。但受版权和加密算法等限制，现在被广泛采用的软件则是 OpenSSH（https://www.openssh.com）。目前 OpenSSH 是 OpenBSD（https://www.openbsd.org）的子项目。很多时候，OpenSSH 常被误认为 OpenSSL（https://www.openssl.org）的子项，但实际上这是两个单独的项目，有着不同的开发团队。但这两个项目有着同样的目标，即提供开放源代码的加密通信软件。

拓展 OpenSSL 是 1998 年开始的开源项目，而 OpenSSH 是 1999 年基于免费的 SSH 1.2.12 版本开发的开源项目。目前，OpenSSH 除了在加密算法的一些数学方法上使用了 OpenSSL 的开源库，其他都已经逐步抛弃了对 OpenSSL 的依赖。但是，在很多 Linux 发行版本中，OpenSSH 还是需要依赖于 OpenSSL。现在的网络设备也越来越多地基于 Linux 平台进行开发，因此在网络设备上使用的 SSH 服务端与客户端也大量地采用了 OpenSSH。

Ubuntu OpenSSH 版本信息：

```
yuxin@Ubuntu:~$ ssh -V
OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8, OpenSSL 1.0.1f 6 Jan 2014
```

Cisco Nexus OS 7.3 OpenSSH 版本信息：

```
switch# ssh -V
OpenSSH_6.2p2, CiscoSSL 1.0.1p.4.13-fips
```

OpenSSH 实现的 SSH 功能除了包括服务端和客户端以外，还包括了一些其他的软件，如密钥的管理和分发等。

网络设备上 SSH 的实现有较大的差异，本节 SSH 的内容侧重于 Linux 平台上的 OpenSSH。

3.2.3 SSH 的基本使用

SSH 现在有两个不兼容的版本，分别是 V1 和 V2。OpenSSH 可以支持两个版本，并且

默认使用 V2 版本。如果要使用 V1 版本，需在参数中单独指定，其参数为 -1。目前只支持 V1 版本的网络设备已非常少见。

最常用的连接网络设备的命令如下：

```
$ ssh admin@10.74.83.166
```

这里，命令中的 admin 是登录设备的用户名。如果这里不指定用户名，那么系统就会使用当前的系统用户名作为登录设备的用户名。另外一个方式是使用 –l 的参数来指定用户名。字符 @ 后就是需要登录的设备的 IP 地址。除此之外，还有一个非常常用的参数，就是指定被登录设备的端口号。

```
ssh -l admin 10.74.83.166 -p 22
```

拓展　使用 username@host 与 -l username host 两个方法有什么区别？这两种方式从最后登录的效果来看是毫无区别的。但是，笔者更加推荐使用参数加值的方式（即后一种方式）。因为，在后期进行程序化处理的时候，后一种方式会更加优雅。

使用过 SSH 登录设备的读者都会知道，当我们第一次登录某台设备的时候，SSH 客户端会获取设备的公钥及其指纹信息。命令的执行结果如下：

```
$ ssh -l admin 10.74.83.166 -p 22
The authenticity of host '10.74.83.166 (10.74.83.166)' can't be established.
RSA key fingerprint is SHA256:SKerR1OHh9tQNMnnXExb7GvJvxA/+x98gx+yc+7UmZQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.74.83.166' (RSA) to the list of known hosts.
```

用户在输入“yes”后，这些信息默认保存在 $HOME/.ssh/known_hosts 文件中。以后再登录这台设备时，SSH 会检查登录的设备和第一次登录的设备是不是一致。如果不一致，则会拒绝登录。其现象如下：

```
$ ssh -l admin 10.74.83.166
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:cbxs75fvW7ZnNu/M6NTEtC+wCxCspmEtlui77WQld+o.
Please contact your system administrator.
Add correct host key in /Users/yuxin/.ssh/known_hosts to get rid of this message.
Offending RSA key in /Users/yuxin/.ssh/known_hosts:7
RSA host key for 10.74.83.166 has changed and you have requested strict checking.
Host key verification failed.
```

在现实操作中，出现这样的问题通常有下面几种情况。

1）被登录的设备重新安装了系统或者是进行了软件升级（软件升级大多不会有问题，

但不排除这种可能性）。

2）设备更换了其他的 IP 地址或主机名，并且更换后的 IP 地址或主机名之前被使用过。

3）登录主机被恶意劫持了，SSH 这样设计的目的是确保每次都登录到相同的设备上。

生产环境使用这样的机制会更加安全，当某台设备被恶意劫持时，管理员通过这样的方式会发现。但如果在实验环境中，这确实带来了很多的不便。如何解决主机指纹检查失败的问题有以下几种方法。

第一种方法：打开 $HOME/.ssh/known_hosts 文件，删除老的记录。也可以使用 sed 命令（具体见下面的代码，命令中的 IP 地址是希望被删除的），关于 Sed 的使用方法，第 5 章会具体介绍。如果 $HOME/.ssh/known_hosts 文件对主机名或 IP 地址进行了散列（HASH），那么这个方式就会失败。

```
$ sed -i -e '/^10.74.83.166/d' $HOME/.ssh/known_hosts
```

第二种方法：使用 ssh-keygen 的命令。-R 的参数用于从 known_hosts 文件中删除所有属于 hostname 的指纹和公钥，这个选项主要用于删除经过散列（HASH）主机的指纹和公钥。

```
yuxin@Ubuntu:~$ ssh-keygen -R 10.74.83.166
# Host 10.74.83.166 found: line 5 type RSA
/home/yuxin/.ssh/known_hosts updated.
Original contents retained as /home/yuxin/.ssh/known_hosts.old
```

第三种方法：在登录时不检查指纹和公钥信息。这个方法应该是最为简单的，但是这样做无疑降低了安全性。命令如下：

```
$ ssh -l admin -o StrictHostKeyChecking=no 10.74.83.166
```

我们可以通过 SSH 扩展工具来管理设备的指纹和公钥。第一种方法、第二种方法都是删除了设备的指纹和公钥，在下次登录的时候再把主机的指纹和公钥保存到 known_hosts 文件中。如何才能通过命令自动更新设备的指纹和公钥？自动更新指纹和公钥的命令如下：

先删除设备的指纹和公钥：

```
$ ssh-keygen -R <IP>
```

通过命令获取设备的指纹和公钥信息，并保存到文件中：

```
$ ssh-keyscan -H [ip_address] >> ~/.ssh/known_hosts
```

通过这两个命令，就可以更新设备的指纹和公钥信息。这种方法应该是最好的管理指纹和公钥的方法。

3.2.4 利用 SSH 远程执行命令

有时候我们只是想在设备上执行一条命令获取一些信息。比如，我们想使用 show version 命令获取设备的版本信息。通常流程是，我们先登录设备，然后输入命令，等待设

备返回结果后，再输入 exit 进行退出。对于 SSH 而言，我们可以不用这样做，只要在 ssh 命令后直接输入需在网络设备上执行的命令即可，如下所示：

```
$ ssh -l admin 10.255.0.80 show ip interface brief vrf management
Warning: Permanently added '10.255.0.80' (RSA) to the list of known hosts.
User Access Verification
admin@10.255.0.80's password:
IP Interface Status for VRF "management"(2)
Interface            IP Address      Interface Status
mgmt0                10.255.0.80     protocol-up/link-up/admin-up
```

我们可以看到，在输入了网络设备的登录密码后，网络设备直接给出了命令执行结果，并且在执行完命令后，退出了 SSH 的登录会话。

除此之外，这些输出内容可以直接使用 Linux 下的文本处理工具进行处理（Linux 这些工具会在第 5 章进行介绍，如果读者之前没有接触过 GREP 工具，可以先跳过）。例如：

```
$ ssh -l admin 10.255.0.80 show version | grep uptime
Warning: Permanently added '10.255.0.80' (RSA) to the list of known hosts.
User Access Verification
admin@10.255.0.80's password:
Kernel uptime is 6 day(s), 4 hour(s), 25 minute(s), 33 second(s)
```

有些读者也许发现了，虽然这样做也许节省了一些时间，但是每次登录设备时都需要输入设备的密码。如果只是执行一次命令还好，但如果需要多次执行命令，这种做法还是很麻烦的。有没有什么好的办法进行优化呢？这里有一个较好的方法可以用来进一步简化操作。

OpenSSH 提供 ControlMaster 功能，在使用 ControlMaster 后，SSH 与设备之间建立一个 Master 连接，之后的所有连接都可以重复使用这一通道，也就是说不管有多少次访问请求，都只需要维护这一个 TCP/IP 连接。后续登录的连接可以不用再输入用户名和密码。

例如：首先，执行如下命令，建立一个 Master 连接。在建立这个连接的时候需要输入密码进行认证。

```
$ ssh -l admin -M -N -f \
-o ControlMaster=yes \
-o ControlPath='~/.ssh/master-%r@%h:%p' \
10.255.0.80
```

然后，后续的命令可以使用如下命令，这个时候就不需要再次输入密码。

```
$ ssh -l admin 10.255.0.80 -o ControlMaster=no \
-o ControlPath='~/.ssh/master-%r@%h:%p' \
show version
```

这样做好像还是很麻烦，每次都需要输入太多的参数。然而，上面这个例子中 -o 后的参数可以配置在 SSH 的配置文件中，这样就不用每次都输入。关于 SSH 的配置见 3.2.5 节的内容。

使用 SSH 远程执行命令的功能，再结合 3.1 节的内容，我们还可以很轻松地使用一条命令恢复一个远程的 screen 会话。这样的命令如何操作，请读者自己思考。我们在本章的小结中会给出例子。

通过使用这个功能，我们可以非常容易地从设备上获取信息，并且在获取信息的时候将不会受到 terminal length 的限制。当读者了解了第 5 章中有关文本处理工具的相关内容后，再结合这个功能就可以实现很多快速处理设备信息的方法。不过，这种方法并不能实现命令交互式的应用环境，比如要修改网络设备的配置。这是因为大部分的网络设备采用交互式的 CLI 来实现此类功能（有一些类型的设备通过一次传递多个命令来实现交互式操作；对于网络设备而言，笔者这里并不推荐采用这样的方式）。

注意 ControlMaster 只支持 OpenSSH 的服务端和客户端，一些老的网络设备并不一定支持这个功能，比如 Cisco IOS 的平台就不支持，而 Cisco IOS-XR/Nexus OS、Juniper JUNOS 等平台可以支持。对于其他厂家的网络设备是否支持这个功能，读者可以自行去测试。

3.2.5 SSH 客户端常用配置

SSH 的配置文件可以在两个地方进行配置。第一个地方是文件 /etc/ssh/ssh_config，这个文件管理本台主机（如 Linux Server）SSH 客户端的配置文件，也就是说无论哪个用户登录这台主机都会使用这个配置。第二个地方是每个用户的目录下，即 $HOME/.ssh/config 这个文件。SSH 客户端配置的内容还是很丰富的，这里我们仅仅介绍一些常用的配置项。

```
1  Host *
2    ControlMaster auto
3    ControlPath ~/.ssh/master-%r@%h:%p
4    StrictHostKeyChecking no
5    UserKnownHostsFile /dev/null
6  Host router1_ios
7    HostName 10.255.0.78
8    User admin
9    Port 22
10   IdentityFile ~/.ssh/yuxin_rsa
```

第 1 行中的 Host * 表示对所有的主机都应用的配置项目。

第 2 行和第 3 行在 3.2.4 节中使用到了。在配置文件中有了这个配置，在命令行中就不需要重复输入那么多的参数。

第 4 行内容在 3.2.3 节中使用过一次，即每次登录的时候不检查 SSH 服务端的指纹和公钥信息。

第 5 行表示把保存 SSH 服务端的指纹和公钥信息的文件指定为一个空文件，这样相当于不保存 SSH 服务端的指纹和公钥信息，这些参数同样可以被写在配置文件中。

接下来的配置文件中出现了 Host router1_ios，这里的 router1_ios 是自己定义的一个主机名，这个名字可以直接在命令中使用。对于后面的几行配置，从名字来看就可以清楚地知道他们的含义了（IdentityFile 指的是私钥的位置）。下面就是使用 SSH 登录这台机器的例子。在例子中，我们可以看到 SSH 软件会在配置文件中查找 router1_ios 的 IP 地址，而不是通过 DNS 获取。由于使用密钥登录，因此不用输入密码。在 3.2.6 节，我们就开始学习如何使用秘钥进行登录和管理秘钥。

```
$ ssh router1_ios
router1#
$ ssh router1_ios show version | grep uptime
router uptime is 2 days, 11 hours, 30 minutes
```

3.2.6　使用密钥登录设备

在上面的例子中，读者应该注意到笔者使用了密钥来登录设备。通过密钥来登录设备最直接的好处是不用再输入密码进行认证，这对于需要频繁登录设备的应用场景来说是有利的。另外，通过密钥对的方式来登录设备可以大大地提高登录的安全性。但随之带来的问题就是密钥的有效管理。

1. 密钥对的产生

使用命令 ssh-keygen 可以产生密钥对。在产生密钥对的过程中，会提示输入一个密码（如果输入了密码而不是空着）。当这个私钥被读取的时候需要提供之前输入的密码，这样就有效地保证了密钥的安全性。读者也许会疑惑，本想取消输入密码的麻烦才选择使用密钥方式进行设备登录，但这里又要加密码岂不是反复输入密码了？暂且放下这个疑问，先来看看如何使用密钥对。一对密钥对分为私钥和公钥（参考 https://zh.wikipedia.org/wiki/ 公开密钥加密），在下面产生的密钥对中，id_rsa 是私钥，id_rsa.pub 是公钥。

```
[root@centos7-1 ~]# ssh-keygen -b 2048 -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
ae:96:47:1e:61:79:4c:a9:84:98:83:b5:ed:b4:0c:71 root@centos7-1
<略>
```

2. 在设备上配置公钥

为了登录设备需要在网络设备上配置（导入）公钥。这里给出了 Cisco IOS 平台和 Juniper JUNOS 平台的配置方法，其他设备的配置请参考厂家提供的文档。

Cisco IOS 配置公钥：

```
router(config)#ip ssh pubkey-chain
```

```
router(conf-ssh-pubkey)#username admin
router(conf-ssh-pubkey-user)#key-hash ssh-rsa <pubic key>
router(conf-ssh-pubkey-data)#end

router#show ip ssh
SSH Enabled - version 1.99
Authentication methods:publickey,keyboard-interactive,password
Authentication Publickey Algorithms:x509v3-ssh-rsa,ssh-rsa
Hostkey Algorithms:x509v3-ssh-rsa,ssh-rsa
Encryption Algorithms:aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,3des-
    cbc,aes192-cbc,aes256-cbc
MAC Algorithms:hmac-sha1,hmac-sha1-96
Authentication timeout: 120 secs; Authentication retries: 3
Minimum expected Diffie Hellman key size : 1024 bits
IOS Keys in SECSH format(ssh-rsa, base64 encoded): router.router.cisco.com
ssh-rsa <公钥字符串，略>
```

JUNOS 配置公钥：

```
[edit system login user yuxin]
user@host# set authentication load-key-file id_rsa.pub
.file.19692 | 0 KB | 0.3 kB/s | ETA: 00:00:00 | 100%
[edit system]
user@host# show
root-authentication {
ssh-rsa "<公钥字符串，略>"; # SECRET-DATA
}
```

3. 使用 ssh-agent 管理密钥

命令 ssh-agent 是用于管理 SSH 私钥的工具，它为私钥提供了长时间且高速的缓存。其通过一个驻留在系统中的进程来实现这个功能。命令 ssh-add 可以将用户需要使用的私钥添加到由 ssh-agent 维护的列表中。之后 SSH 需要使用私钥登录设备时，会优先在这里寻找私钥的高速缓存部分。命令如下：

```
$ eval 'ssh-agent'
Agent pid 53053
$ ssh-add yuxinpw_rsa
Enter passphrase for yuxinpw_rsa:
Identity added: yuxinpw_rsa (yuxinpw_rsa)
```

eval 后面使用的是反引号（`），位于键盘波浪号（~）下面，实际为同一个键。对于美式键盘，其位于数字 1 前面的那个键。

命令 ssh-add 添加了带有密码的私钥，在添加的时候就会提示输入密码。以后在 SSH 中使用这个私钥的时候就不需要再一次提供密码。

对于前面的疑问，这里终于得到了解释，带密码的私钥仅仅在添加到密码库时被要求输入密码，而登录设备时将不再需要输入密码。

```
$ ssh-add -l
2048 9b:0a:d4:2c:00:fb:69:59:e4:6f:a5:ca:af:8e:cc:aa .ssh/yuxin_rsa (RSA)
1024 aa:50:a5:a4:14:74:27:db:89:14:e8:e8:c9:55:41:38 yuxinpw_rsa (RSA)
```

使用命令 ssh-add -l 可以查询现在 ssh-agent 管理了哪些私钥。

通过上述的方法可以让设备的登录过程更加安全，同时提升了登录设备的便利性。

注意　对于一个用户，使用公钥认证的方式和使用 tacacs+ 或 radius 的认证方式不能同时使用。要么使用公钥认证，要么使用密码认证（密码认证可以使用 tacacs+ 或 radius）。公钥认证用于本地认证，而 tacacs+ 或 radius 的认证方式用于集中认证。不过，无论通过哪种认证方式，都可以使用 SSH 进行登录。如果设备支持 ControlMaster 的功能，将可以简化登录过程中频繁输密码的过程。

3.2.7　使用 scp 进行文件传输

命令 scp 是 secure copy（安全复制）的简写，用于进行远程复制文件。它与远程设备通信的通道是和 SSH 一样的，加密方式也是类似的。其传输端口和 SSH 是一样的，默认为 tcp 22。

1. 基本命令和参数

```
$ scp [参数] [原路径] [目标路径]
$ scp admin@router1_ios:/config/vios vios
```

远程设备的路径表示为 username@hostname:path，用户名和远程设备的名称或 IP 用“@”进行分割，这点和 SSH 登录方式是一致的。远程设备上的路径和设备名称中间用“:”进行分割，建议在这里写出远程设备的绝对路径。

下面列出几个常用的参数。

- -q：不显示传输进度条，常用于脚本中。
- -r：递归复制整个目录，这在复制多个文件时很有意义。
- -l：limit，限定用户所能使用的带宽，以 kbit/s 为单位。
- -P：port，这里是大写的 P，port 是指定数据传输用到的端口号。

2. 常见网络设备的配置

对于服务器而言，通常来说使用 OpenSSH 的服务端就默认支持了 scp 的文件传输方式。但是对于网络设备而言，这个功能往往并不是默认开启的，需要我们进行额外的配置。具体到不同的网络设备会有一些差异，下面给出几款网络设备的例子。

Cisco IOS 设备：在配置了 SSH 登录之后，使用如下命令开启 scp 服务。

```
router(config)#ip scp server enable
```

Cisco Nexus 设备：同样和 IOS 类似，在配置了 SSH 登录之后，使用如下命令开启 scp

服务。

```
switch(config)# feature scp-server
```

Juniper JUNOS：在 JUNOS 上只需要开启 SSH 服务就同时开启了 scp 的服务。

```
user@host# set system services ssh
```

3.2.8 利用 SSH 端口隧道转发功能

正如前面介绍，SSH 可以用于加密的远程登录、文件传输等功能。除此以外，SSH 还有一个非常有用的功能，就是端口隧道转发功能。我们熟悉的 POP3、SMTP、FTP、LDAP 等协议都可以利用此功能，通过 SSH 的加密隧道进行数据传输，从而弥补其传输协议不加密的缺点。除了提供通道的加密功能，SSH 还能完成端口的转发功能（转换目的或源端口）。我们先看一下图 3-4。在图 3-4 中，网络管理服务器通过交换机直连到了路由器的管理口。堡垒机和网络管理服务器之间有一层 NAT 设备，堡垒可以直接访问网络管理服务器，但是网络管理服务器不能主动访问堡垒机。堡垒机上也没有路由器的管理网段路由。基于传统的登录方式，堡垒机必须先使用 SSH 登录到网络管理服务器，然后才能通过网络管理服务器再登录到路由器。如果要给路由器传文件，也必须先从堡垒机拷贝到网络管理服务器，然后拷贝到路由器上。这样的运维方式是非常烦琐的。在这种场景下，我们可以使用 SSH 的端口隧道转发功能来简化此烦琐流程。

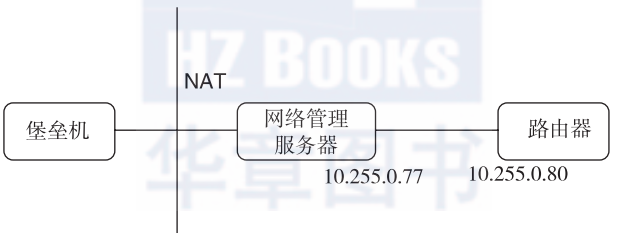


图 3-4 网络管理拓扑

先在**堡垒机**器上执行如下命令，完成本地端口转发以及建立 SSH 隧道。

```
$ ssh -f -N -L 10080:10.255.0.80:22 network_mgt
```

说明如下。

- -f：认证结束后在后台运行此命令，通常和 -N 一起使用。
- -N：不执行 shell 脚本或命令，通常和 -f 一起使用。
- -L：创建一个本地转发规则。将本地（这里指堡垒机）的某个端口（本例中为 10080）转发到远端指定设备（本例中为路由器的 IP：10.255.0.80）的指定端口（例中为 SSH 的默认 TCP 22 端口）。其工作原理是：本地机器分配一个端口，一旦这个端口有了数据，该规则就经过安全通道转发到远程主机的端口。从图 3-4 可以看

出，这里利用了堡垒机和网络管理服务器之间的 SSH 连接建立了一个 SSH 的隧道（IPv6 地址用另一种格式说明：port/host/hostport）。

- network_mgt：连接到网络管理服务器的参数。其更加完整的形式如下：

```
-l yuxin 10.74.82.252 -p 10000 -i ~/.ssh/yuxin_rsa
```

因为在 $HOME/.ssh/config 中有如下配置，这里只用了 network_mgt。

```
Host network_mgt
    HostName 10.74.82.252
    Port 10000
    User yuxin
    IdentityFile ~/.ssh/yuxin_rsa
```

我们可以通过如下命令来查看这个转发策略是否已经运行。

```
# ps aux | grep -i "ssh -f"
root 29768 0.0 0.0 73940 1104 ? Ss 15:03 0:00 ssh -f -N -L 10080:10.255.0.80:22 network_mgt
```

现在我们可以在堡垒机上直接通过本地的 10080 端口直接登录到远端的路由器或交换机。

```
$ ssh localhost -l admin -p 10080
User Access Verification
admin@localhost's password:
    Cisco NX-OS Software
Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.
<略>
switch#
```

也可以直接运行远端设备上的命令。

```
$ ssh localhost -l admin -p 10080 show system uptime
User Access Verification
admin@localhost's password:
System start time:          Sun May  7 09:32:40 2017
System uptime:              7 days, 3 hours, 26 minutes, 2 seconds
Kernel uptime:              7 days, 3 hours, 27 minutes, 37 seconds
```

还可以通过 scp 来传递数据。

```
$ scp -P 10080 admin@localhost:/scripts/maintenance-mode.py maintenance-mode.py
User Access Verification
admin@localhost's password:
maintenance-mode.py                   100%   12KB  12.0KB/s   00:00
```

上面端口隧道转发的方法对网络设备并没有什么特殊的要求，即使网络设备不支持 SSH。所有功能都是在堡垒机和网络管理机之间完成的。

现在我们演示一下：先在网络设备上开启 Telnet 服务，然后在**堡垒机**上执行如下命令。

```
$ ssh -f -N -L 10083:10.255.0.80:23 network_mgt
```

通过 Telnet 登录设备。

```
$ telnet -l admin localhost 10083
Connected to localhost.
Escape character is '^]'.
Password:
Last login: Sun May 14 13:12:32 UTC 2017 from 10.255.0.77 on pts/0

Cisco NX-OS Software
Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.
<略>
switch#
```

在这个例子中，我们可以看到 Telnet 服务也使用了 SSH 的加密通道。

除了本地端口转发模式，SSH 还有远程端口转发模式，需要把 -L 的参数改成 -R。具体的细节希望读者自己去研究学习，这里将不再举例。

3.2.9 利用 SSH 做 Socket 代理

3.2.8 节提到的端口隧道转发功能是静态的端口的，本节介绍动态的端口隧道转发功能。这里，还是基于图 3-4 中的拓扑图。假设现在路由器是一台能提供 Web 服务的设备，我们可以通过浏览器来管理这台设备。这时，我们通常的做法也许会在网络管理服务器所在的网络（网段）中安装一台 Windows，然后通过远程桌面登录这台 Windows 服务器，运行 Windows 服务器上的浏览器来管理这台路由器。这样做显然是非常麻烦的，你完全可以换一种方式来实现。

首先，在堡垒机上运行如下命令：

```
# ssh -f -N -g -D 10088 network_mgt
```

查看端口服务情况：

```
# netstat -natp | grep 10088
tcp        0      0 0.0.0.0:10088           0.0.0.0:*               LISTEN      30041/ssh
tcp6       0      0 :::10088                :::*                    LISTEN      30041/ssh
```

我们可以看到，在堡垒机上 TCP 10088 端口提供了一个服务。这个服务可以通过堡垒机和网络管理服务器之间的 SSH 隧道访问到路由器。这时候，我们在一台可以正常访问堡垒机 TCP 10088 端口的计算机上使用浏览器来直接访问路由器的 Web 服务，不过我们需要在浏览器上设置一个 Socket5 代理。对于 FireFox 浏览器，我们可以在连接设置中找到这个 Socket5 代理服务（见图 3-5）的其他浏览器，也可以查找其设置的位置，或者使用第三方的插件来完成相应的功能。

3.3 小结

本章主要介绍了 screen 和 SSH 两个工具，这两个工具主要用于管理会话和登录网络设

备。对于广大的网络工程师，SSH 应该不是一个陌生的工具。通过本章一些例子，大家可以了解和掌握 SSH 相关的一些其他功能。通过使用这些功能，相信能够给大家日常的工作带来更多的方便。

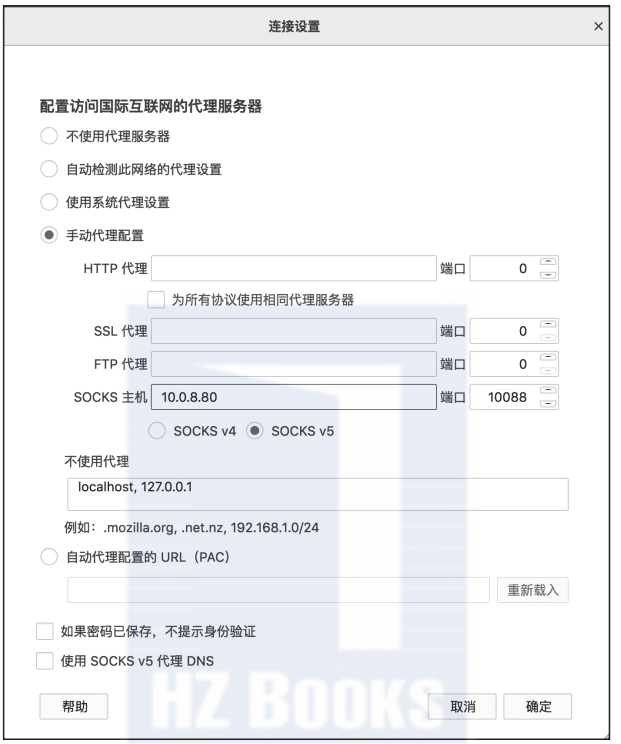


图 3-5　FireFox Socket 5 代理

下面来看一下在 3.2.4 节中留下的思考内容。

首先，使用如下命令查询其他机器的 screen 会话。

```
$ ssh -l root 172.16.6.130 screen -ls
root@172.16.6.130's password:
There are screens on:
    29645.mgt      (Multi, detached)
    23319.pts-5.centos7-1 (Multi, detached)
    22596.yuxin    (Multi, detached)
Sockets in /var/run/screen/S-root.
```

然后，使用如下命令恢复远程 screen 会话。

```
$ ssh -t -l root 172.16.6.130 screen -r mgt
```

必须添加 -t 的参数。

在这个 screen 会话中，使用 C-a d 命令分离时，会直接退出 SSH 的连接。