HZ BOOKS
华章科技

Linux/Unix
技术丛书

Puppet
权威指南

王冬生 著

Puppet：The Definitive Guide

- 腾讯高级运维工程师撰写，腾讯、百度、阿里巴巴、新浪等多家企业总监级运维专家高度认可并联袂推荐
- 从基本功能、操作使用、高级功能、二次开发、工作原理、性能优化、疑难问题等多个角度系统、深入讲解了Pupept运维的方法、技巧和最佳实践，包含多个企业级实战案例

机械工业出版社
China Machine Press

Puppet
权威指南

Puppet The Definitive Gwide

王冬生 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Puppet 权威指南 / 王冬生著 . —北京：机械工业出版社，2015.1
（Linux/Unix 技术丛书）

ISBN 978-7-111-48598-8

I. P… II. 王… III. 程序开发工具 IV. TP311.52

中国版本图书馆 CIP 数据核字（2014）第 273770 号

自动化运维领域的重磅之作，Puppet 领域权威的、系统的、有深度的、极具企业实战价值的著作。腾讯高级运维工程师撰写，获得了来自腾讯、百度、阿里巴巴、新浪等多家世界级互联网企业的总监级运维专家的高度认可和联袂推荐。

工欲善其事，必先利其器，在操作层面，本书从 Puppet 的基本功能到高级功能，再到二次开发均有详细分析，内容详尽而系统，能帮助读者全面掌握 Puppet 的使用；知其然，更要知其所以然，在原理层面，本书从多个层面和角度分析了 Puppet 的工作原理，能让读者更深入理解和使用 Puppet；好的经验是无价的，在应用层面，作者将自己几年来学习和应用 Puppet 积累的方法、技巧、最佳实践以及解决疑难问题的秘诀都毫无保留地奉献了出来，能让读者少走弯路，事半功倍；实践出真知，在实战层面，本书给出了几个对企业很重要的、常见的综合性案例，不仅能帮助读者提高实战能力，还能给予他们解决这些问题的良好解决方案。

Puppet 权威指南

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：孙海亮
责任校对：董纪丽
印　　刷：北京市荣盛彩色印刷有限公司
版　　次：2015 年 1 月第 1 版第 1 次印刷
开　　本：186mm × 240mm 1/16
印　　张：23
书　　号：ISBN 978-7-111-48598-8
定　　价：69.00 元

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

Foreword 序 一

如何成为一名优秀的运维工程师？利用哪些工具和手段去协助自己？这些问题的解答不仅适用于专业人士，也适用于初学者，而本书就是一本技术人员必备的良书。本书面向的读者是那些对技术有着执著热情的运维和开发人员，书中对如何成为一名优秀的运维工程师，如何使用自动化工具 Puppet，以及 Puppet 原理、部署等方面都做了详细介绍。我相信这本书能给所有初入技术领域的人一个整体框架，也能给专业的技术人员一些思路和细节上的指引。

我们周围有很多的运维人员都在踌躇：该如何成为一名优秀的运维工程师？放在七八年前，在那个互联网并未发展到如今这般庞大的时代，在一切仅仅靠人脑就能实现运维的机器规模下，在靠简单的自动化程序就能完成运维的状态下，运维工程师只要维护好 Server，保障应用稳定、不出故障，也许就能达到一名出色的运维工程师的标准。但是现在，在这个大数据的时代，原有的标准已经无法满足需求。一名出色的运维工程师除了做到维护系统稳定和不出故障外，还要参与系统架构的设计，做到大规模机器下的自动化安装、部署，高并发、高压力下的性能调优，符合诸多因素，才能成为一名合格的运维工程师。

本书不仅内容详尽，而且结构清晰，使用了大量实例进行详细的表述和分析，便于读者理解及查阅，具有很强的实用性和指导性，扩展了读者对运维概念的认知。

高国新　阿里巴巴技术保障－云 PE、技术专家

序 二 *Foreword*

相传，在很久很久以前，遥远的西方国度中，一家叫雅虎的网站，在普天下的系统工程师心中埋下了一颗神奇的种子，从此变得一发不可收拾。围绕着这颗种子，总会有讲不完的传奇故事，令无数人想一窥究竟；而它，也承载了数不清的美好憧憬。这颗种子，就是自动化运维，一个永恒的话题。

回顾过去 10 年的互联网从业历程，我有幸亲眼见证了自动化运维在中国所发生的一系列深刻变化，但同时在这方面也深存遗憾。遗憾源于自己曾亲手规划过一套自动化运维的美好蓝图，但最终变为黄粱一梦。坦白地讲，在自动化运维方面，我的失败教训远大于成功经验，今天在这里写序，心中难免有一丝不安。尽管时隔多年，但对那段失败经历的复盘和自我反思，在我心中从未停止过。以今天的视角看，我认为自动化运维涉及“期望管理、资源整合、冲突管理、工具选择、风险管控、思维理念的塑造”等很多方面，甚至还需要充分理解公司的战略规划。

就技术层面而言，我想结合本书要讲的 Puppet，谈一下自己对“运维工具选择”的理解。自 Puppet 诞生之日起，它就如同夜空中最闪亮的那颗星星一样，一举超越了那个叫 Cfengine 的“老前辈”，引起了无数系统工程师的注目和追随。直到今天，它的影响力和全球部署量依然很大。但同时有关 Puppet 是否过时的讨论也悄然兴起。在我看来，一方面，世间万物都有各自的生命周期，Puppet 当然也不例外。与其讨论它是否过时，它何时会过时，倒不如全面而深入地理解 Puppet 的思想精髓，深入思考它在未来可能会面临的挑战，以及 Puppet 本身在版本迭代过程中是如何应对这些挑战的。另一方面，与其纠结于目前是该用 Puppet 还是直接上线 SaltStack，倒不如深入思考：我们的自动化运维框架如何能更快速而低风险地应对上层工具可能发生的变化。

而本书的价值恰恰在于，不仅用全景式的视角对 Puppet 的使用进行了翔实的介绍，更有对细节的深度解读。透过这些深度解读，读者朋友们可以逐步悟出 Puppet 的思想精髓，进而对自动化运维有更深的理解。书中所配的具体案例，都是作者多年来在多家大型互联网公司关键岗位工作所得精华之总结，具有非常强的权威性和实操性。

除对 Puppet 的讲解外，我本人也非常喜欢本书的第 1 章。这与我前文提到的自动化运维所涉及的“思维理念的塑造”不谋而合。相信大家读完这本书后，一定会觉得物有所值！

李晓栋　新浪网研发中心高级技术经理

前 言 *Preface*

为什么要写这本书

早在2009年的时候，笔者就梦想能出版一本属于自己的书，虽然那时用业余时间写过一本，但并不专业也没有正式发行。时隔5年，随着工作经验的积累和知识的增长，经过了2年多的构思，编写的这本书终于面市了，这算是圆了笔者一个小小的梦想。笔者从事互联网行业已有7年左右的时间了，与传统行业相比，7年的时间仅能算是初出茅庐，但对于年轻的互联网行业来说不算短了，因为互联网在1995年才正式进入中国，目前还处于“花样年华”阶段。

2008年～2011年5月，笔者就职于新浪网技术中国有限公司。当时笔者所在的团队主要负责登录系统的开发与运维，同时还负责包括好友系统、在线系统和消息系统的维护，所有服务器加在一起有100多台。除了不用到机房上线服务器与安装系统外，其他的所有工作都由笔者当时所在的团队共同完成。那时都是由开发工程师兼运维工程师来完成这100多台机器的维护、软件安装、程序部署、架构部署、系统调优与监控等工作。这样的工作方式有优点也有缺点。优点是一名初出茅庐的职场新人（笔者）就可以自己建立运维工具，构建系统架构和部署系统的监控，并且从中获取更多的知识，迅速成长；缺点是团队中的同事都有自己的一套独立的小工具，团队中的工具与工具之间不能继承与复用，每次系统增加新功能都需要重新开发工具，浪费了很多人力成本。而且工具本身不够自动化，很多环节需要人为参与。那时笔者就在思考，是否能有一种工具来帮我们串联贯通业务，在节约人力成本的同时提供个性定制操作呢？

2011年6月至今，笔者就职于腾讯。在腾讯工作的这几年中，笔者专职从事运维与运维工具的开发等工作。笔者将目前在腾讯的运维经历划分为以下4个阶段。

- 第一阶段，**统一化**：即操作系统发行版本都是统一的，服务器硬件也采用的是相同型号、相同配置，从根本上规避了差异化给运维工作带来的麻烦。

- 第二阶段，**基础化**：从系统安装、上线、服务部署、测试、监控到后续维护均由统一的系统完成。
- 第三阶段，**自动化运维**：服务器软件安装和更新，程序发布版本都由相应工具完成，不需要人为参与。通过 Web 化与图形化的展示规避了服务器误操作的风险，提高了系统可用性与工作效率。
- 第四阶段，**大数据挖掘**：新产品上线需要多少台服务器、带宽和流量仍处于不好估算的阶段。当系统上线后，要想通过海量数据发现产品潜在的问题、获悉用户爱好与习惯、预测未来产品走势，就需要我们了解与掌握相关信息，所以需要通过大数据的分析，从产品海量日志中找到有价值的信息，助力产品更好地发展。

当很多公司的运维还处于第二与第三阶段时，腾讯已经逐渐从第三阶段步入第四阶段。但在这 4 个阶段中，笔者觉得第三阶段是一个非常重要的阶段，只有解放了人力才能将更多的精力投到大数据挖掘上，以助力产品发展。2011 年下半年，笔者偶然听说了 Puppet 配置管理工具很不错，并从官方文档上了解到，Puppet 配置管理工具正是笔者一直在找寻的工具。它安装简单，使用方便，并且有着丰富的文档与活跃的社区，这无疑降低了初学者学习门槛。所以在对 Puppet 进行了系统的了解与学习之后，恰好赶上一次新业务上线，部分功能需要运营系统再次开发支持，笔者觉得这是一次测试、使用 Puppet 的好机会，于是在一周内搭建并使用了 Puppet 配置管理工具。测试表明，在无需人力参与开发的情况下，Puppet 配置管理工具确实节约了很多人力成本，提高了效率。2013 年 6 月，笔者决定把学习与使用 Puppet 配置管理工具的过程和心得整理成文字并发表出来，以让更多的计算机爱好者和工作者了解并使用这款软件。编写本书的过程也是再次学习 Puppet 的过程，Puppet 并不只是配置管理工具，它的设计思想与软件架构，以及版本迁移历程都值得我们去学习与借鉴。笔者衷心希望本书能帮助读者更快地掌握 Puppet 管理配置工具，并将其应用到实际工作中，为大家带来方便。

本书主要内容

本书共分为 18 章，4 个部分。

第一部分　基础篇（第 1 ～ 5 章）：第 1 章对比了目前常见的自动化运维工具，并介绍了目前应用 Puppet 的公司与 Puppet 发展前景，让读者了解为什么选择 Puppet，Puppet 与其他运维工具相对而言有哪些优势，使大家对 Puppet 有个基本的了解与认识。第 2 ～ 5 章主要介绍 Puppet 的版本分支状况及选择，Puppet 安装过程、目录结构、各版本之间命令差异如何解决，Puppet 配置文件的作用等。基础篇学完后，读者可以搭建 Puppet 环境并掌握基本使用方法。

第二部分　进阶篇（第 6 ～ 9 章）：主要介绍 Puppet 核心编程语言、资源、模板应用与 Facter。让读者能够在搭建的基础上完全玩转 Puppet。

第三部分　高级篇（第 10 ～ 15 章）：主要介绍 Puppet 的一些高级功能。当 Puppet 不能满足我们的工作需要时，如何做二次开发使其能够为我们工作所用？大规模使用 Puppet 时，性能瓶颈应该如何解决？如何管理与查询差异化服务器信息及上报的日志？海量的 Agent 服务器中部分 Agent 工作异常如何快速定位原因？这些都是在 Puppet 实际使用中常常会遇到的问题，读者们可以在本篇中找到答案。

第四部分　应用篇（第 16 ～ 18 章）：这几章会以案例形式介绍 Puppet 在企业环境中如何应用，在方便读者记忆的同时，拓展读者的思路。对 Puppet 的了解和使用达到一定程度后如果读者想偷懒，还可以使用热心网友分享的、已经写好的 Puppet 配置语言，这样可以将更多的时间放在系统优化与数据挖掘上。

本书特色

- 在读本书前并不要求读者对 Puppet 配置管理工具有一定使用经验或者编程语言的背景。本书从零基础开始介绍 Puppet 配置管理工具以及其管理配置语言，通过由浅入深的案例让读者更好地掌握 Puppet。
- 本书在写作过程中主要参考 Puppet 两个稳定的版本分支文档，即 Puppet 2.7（https://docs.puppetlabs.com/puppet/2.7/reference/）和 Puppet 3.6（https://docs.puppetlabs.com/puppet/3.6/reference/）。
- 笔者将 Puppet 引入工作环境，不仅是为了解与学习 Puppet，更是要解决工作中遇到的实际问题。书中案例贴近实际工作环境，同时在不同的工作场景下给出了不同的使用建议供读者选择。
- 本书不仅系统地介绍了 Puppet 配置管理工具，还介绍了一些 Puppet 辅助工具，如 PuppetDB、Marionette Collective、DNS 和版本配置工具。

读者对象

- 系统管理员、运维工程师、网络管理员
- 自动化运维工具爱好者
- 开源软件爱好者

对读者的建议与意见反馈

本书出版时 Puppet 版本已经升级到了 Puppet 3.6，但本书案例主要使用的是 Puppet 2.7 版本，需要读者注意的是两个版本间小部分命令的使用可能会有所不同，但差异并不大，并不

影响我们在版本过渡期间的使用。如果读者刚接触 Puppet，对 Puppet 还不是很熟悉，笔者建议温故而知新，先从 Puppet 2.7 版本入手再逐渐升级到 Puppet 3.6 版本上，这样才能更好地掌握 Puppet。另外，如果读者已经使用过 Puppet，并希望通过本书获取更多的个性化的内容并应用到自己的工作中，笔者建议通过二次开发来改造 Puppet，从而为我们的工作所用。因为 Puppet 是一款开源软件，它并不能解决我们工作中遇到的所有问题，但它的优势是提供了很多 API 和程序源码供我们更方便地改造它。

笔者在编写本书的过程中也遇到了很多困难，并通过网上查找、与朋友交流、测试和自己理解一一克服。由于时间仓促，本书依然难免会有错误或不准确的地方，希望读者能够指正，以求一同学习进步。如果读者对本书有任何意见或发现任何错误，请和笔者联系，可以将意见发送到邮箱 7717060@sina.com，标题注明“《Puppet 权威指南》”，笔者将会认真查看大家的批评和建议。同时笔者也开通了个人博客 www.puppeter.com，本书的勘误、代码下载与后续更新都会在个人博客上与大家分享。另外本书涉及的源代码也可到华章公司官网（www.hzbook.com）下载。

致谢

在本书出版的过程中得到了很多朋友与家人的支持。

首先感谢华章出版社的孙海亮与杨福川。在本书编写的一年中，编辑孙海亮耐心地审阅并指出了笔者的很多错误与不足，同时给出了很多专业性的建议，他是一位非常称职的编辑，值得敬佩。杨福川给予了笔者更多的鼓励与资源的支持。

特别感谢我的朋友与同事赵建春、梁定安、李晓栋、王炜煜、于杰英和高国新对本书的建议与支持。

感谢公司的同事陶凛然、吴伟彬、黄浩宇、黄伟俊、张兰、黄兆鹏、李平安、赵子青、李金榜、王秀才、杨波、曹凤龙、聂鑫、刘华、张志谭、余东良和孙凯荣在软件环境、使用与测试方面对本书的帮助。

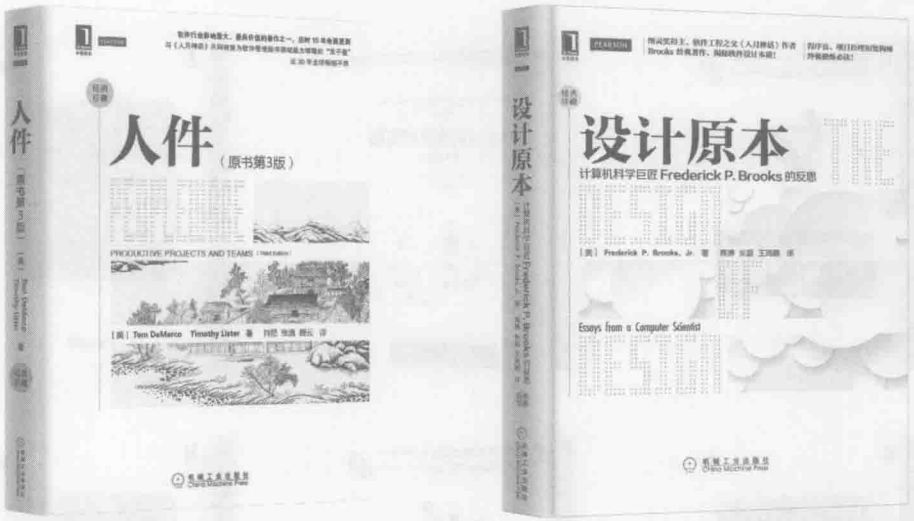
还要感谢刘志宇、吴城、汪洋、周荣茂，他们在本书后期推广方面给出了很多建议与帮助。

最后感谢我的老婆，是她默默地在后方支持着我，促使我最终能完成本书。

王冬生

于深圳南山科技园

推荐阅读



人件（原书第3版）

作者：（美）Tom DeMarco 等 ISBN：978-7-111-47436-4 定价：69.00元

公认对软件行业影响最大、最具价值的著作之一，历时15年全面更新
与《人月神话》共同被誉为软件图书领域最为璀璨的“双子星”，近30年全球畅销不衰

在软件管理领域，很少有著作能够与本书媲美。全书从管理人力资源、创建健康的办公环境、雇用并留用正确的人、高效团队形成、改造企业文化和快乐工作等多个角度阐释了如何思考和管理软件开发的最大问题——人（而不是技术），以得到高效的项目和团队。

设计原本——计算机科学巨匠Frederick P. Brooks的反思（经典珍藏）

作者：（美）Frederick P. Brooks,Jr. ISBN：978-7-111-41626-5 定价：79.00元

图灵奖得主、《人月神话》作者Brooks封笔之作，揭秘软件设计神话！
程序员、项目经理和架构师必读的一本书！

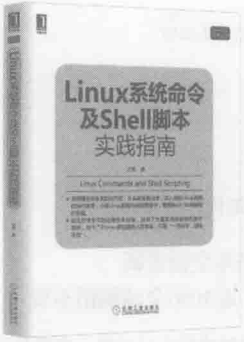
《设计原本》开启了软件工程全新的“后理性时代”，完成了从破到立的圆满循环，具有划时代的重大里程碑意义，是每位从事软件行业的程序员、项目经理和架构师都应该反复研读的经典著作。全书以设计理念为核心，从对设计模型的探讨入手，讨论了有关设计的若干重大问题：设计过程的建立、设计协作的规划、设计范本的固化、设计演化的管控，以及设计师的发现和培养。

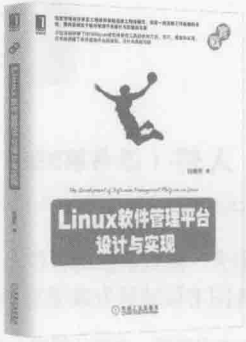
推荐阅读

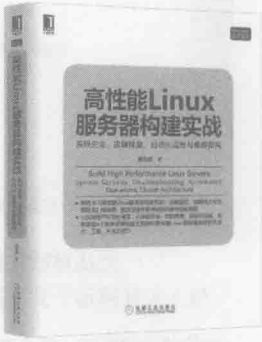

Puppet实战


Nagios系统监控实践


Python
自动化运维
技术与最佳实践

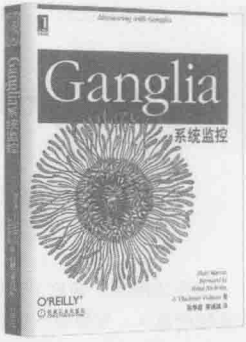

Linux系统命令
及Shell脚本
实践指南


Linux软件管理平台
设计与实现


高性能Linux
服务器构建实战


高性能Linux服务器构建实战
运维监控、性能调优与架构应用


构建高可用
Linux服务器


Ganglia
系统监控
O'REILLY

目 录 *Contents*

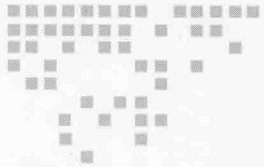
第二部分 进 阶 篇

第三部分 高 级 篇

第一部分 Part 1

基 础 篇

- 第1章 运维工程师的利器——自动化运维工具
- 第2章 Puppet介绍
- 第3章 Puppet及相关工具的配置与安装
- 第4章 Puppet目录结构、配置文件和命令详解
- 第5章 通过Puppet构建主机



Chapter 1 第 1 章

运维工程师的利器——自动化运维工具

随着网络云时代和大数据时代的到来，运维工程师负责管理的服务器数量也成倍地增长。如何管理好这些服务器为云时代和大数据时代保驾护航，是摆在运维工程师面前的一道难题。而解决这道难题就需要运维工程师对自动化运维工具的掌握达到一定的程度。笔者希望通过本章抛砖引玉，结合自己的经验介绍多年来使用自动化运维工具的心得和体会。本章首先介绍互联网运维工程师的职责、优秀运维工程师和普通运维工程师的区别；然后简要介绍常见的自动化运维工具；最后比较当前常见的自动化运维工具的优势，以此说明为什么要选择 Puppet 做我们的自动化运维工具。

1.1 浅谈运维工程师

想必大家都看过《好的程序员是普通程序员效率的数十倍》这篇文章，这句话是比尔·盖茨说的，被很多文章引用和转载。笔者读后感同身受，觉得这篇文章讲的并不夸张。程序员如此，运维工程师也是如此，一个优秀运维工程师的效率确实是普通运维工程师的数十倍。本节笔者将带领大家了解一下优秀运维工程师和普通运维工程师之间的不同之处。我们从运维工程师的定位和职责开始介绍，继而详细分析普通运维工程师和优秀运维工程师的差别，最后落脚到自动化运维工具。

1.1.1 运维工程师定位和职责

要想了解普通运维工程师和优秀运维工程师之间到底有什么不同，首先要了解运维工程师的定位，也就是运维工程师处于工作链的哪一环节；然后要了解运维工程师的职责是

什么。下面笔者将分别进行介绍。

1. 定位

首先我们来看运维工程师处于工作链的哪一环节。我们都知道，公司的老板需要根据市场提出指导思想；项目经理需要对市场状况进行调研分析；产品经理需要根据老板的指导思想，结合项目经理的调研分析设计出产品雏形并提交给开发工程师。在此期间，系统工程师和网络工程师需要根据产品选择 IDC、搭建网络环境、分配网络相关资源；开发工程师编写代码，并将开发后的产品提交给测试工程师；测试工程师对产品做系统和功能测试，将稳定的代码提交 SVN，这时轮到运维工程师上场了——运维工程师将代码从 SVN Check Out 出，推到线上系统供用户访问，并进行后续的升级维护等工作。从图 1-1 不难看出，运维工程师处于最后一个环节，也是最重要的一个环节，运维工程师负责产品后期的维护、升级、监控和服务保障等。如果运维工程师在日常运维中工作做得不到位就直接影响线上用户的访问和体验，后果是比较严重的。

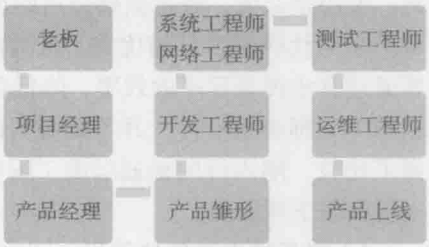


图 1-1　产品研发流程图

需要注意的是，不同行业的公司对运维工程师的定义也不同。笔者在这里是以互联网公司为背景介绍的，当然即使同样处于互联网行业中的公司也会存在一些差别。规模小一点的公司，一个运维工程师可能需要完成包括系统工程师、网络工程师、测试工程师及部分开发工程师的工作内容；规模比较大的互联网公司，运维工程师的工作职责可能会比笔者介绍的更细致些。

2. 职责

有关运维工程师的职责，我们在这里同样以互联网公司为例。笔者将整个运维工作的链条细化为以下 3 个部分。

- 基础运维：硬件选型、IDC 部署、架构规划、容量规划、资源成本管理、预算控制、容灾、高可用设计和架构可扩展性等。
- 业务运维：发布变更、监控告警、故障处理、软件安装升级、系统调优、运营质量报告、线上环境备份、运营大数据分析、恢复和线上程序发布维护等。
- 数据运维：存储架构设计、提升数据的均衡负载能力、数据的备份与恢复、容灾的建设、数据的冷热分离、监控告警、DB 调优与 SQL 优化等。

1.1.2　优秀运维工程师 vs 普通运维工程师

优秀运维工程师和普通运维工程师确实存在着一些差别，特别是在一些大的互联网公司内，一个优秀的运维工程师在带来稳定服务的同时还可以为公司节约可观的成本。本小

节笔者结合自己的工作经历和体会，来分析一下优秀工程师和普通运维工程师究竟存在哪些差别。笔者希望通过本小节的介绍能使所有的运维工程师找到一点儿成为优秀运维工程师的灵感。当然很多运维工程师已经很优秀了，但是没有最好只有更好，我们可以综合其他优秀运维工程师的优点，让自己变得更优秀。

在《好的程序员是普通程序员效率的数十倍》中有这样一段话："很多程序员每天编码超过2000行，对于一个程序员来说这很正常，但我们可以说这是一个好的程序员吗？我们再来看一下，网上报道著名的软件公司——微软公司，他们的程序员每天只编写50行代码，差距这么大，难道微软公司的员工就不是好的程序员？"从这段话来看，程序员并不是每天写的代码越多就越优秀。优秀的程序员能够写出正常运行的程序是基本要求；但还需要通过算法提高程序的效率，降低程序使用的系统资源，同时提高程序的复用性和易用性。运维工程师也是一样，并不是每天的工作量大就能成为一个优秀的运维工程师。笔者自己做了比较，刚入行的普通运维工程师和工作多年的优秀运维工程师有着明显的差别。差别主要在以下两方面。

1）**对运维工具集的使用。**刚入行的运维工程师习惯自己写脚本来完成运维工具建设和日常的运维工作。笔者当年也一样，每次接到新任务都会重新写一套脚本来完成新项目。笔者在工作多年后，也时常看到很多刚入行的运维工程师采用和笔者当年一样的工作方式，这样无疑是比较耗费时间的。单就这方面来说，优秀运维工程师和普通运维工程师相比，主要有以下特点。

- 撰写通用的脚本。优秀运维工程师会找到每个项目的共同点写出通用的脚本，尽量降低重复劳动，提升工作效率，同时将通用的脚本共享。这样不仅降低了其他人的工作成本，还提高了整个组或部门的工作效率。
- 尽量使用开源工具。开源工具本身就节省了开发的成本。如果开源工具不能满足项目需求，优秀运维工程师会对开源工具做二次开发。这样一方面节约了大量的成本，另一方面他们也可以借鉴开源工具的一些思路和想法来拓展自己的视野。

2）**时间管理。**在互联网行业中，运维工程师和其他工种有很大的差别。运维工程师的时间比较零散，比如他们可能前一秒钟还在升级线上程序，后一秒钟由于程序BUG导致线上服务不可用需要他们立即停止正在进行的工作，准备迁移线上流量。运维工程师好比一个消防员，每天处理的都是十万火急的问题，一点都疏忽大意不得。而这样忙碌地工作一天后，总结起来可能实质性的工作并没有很多。其实这是很多运维工程师的弊病。所以想要成为一个优秀运维工程师，就要合理地管理和利用时间。好比一个球队在踢球前会预先设置好阵形，然后在比赛的过程中通过变换阵形来打赢比赛。运维工程师的工作也是如此，需要合理分配时间，平日可以用70%的时间来处理日常的运维工作，20%的时间通过程序或工具将现有流程实现自动化，用10%的时间来提前准备明天的工作。只有这样合理分配时间，才能让我们的日常运维工作事半功倍、井井有条。

除了上面提到的对运维工具集的使用和时间管理外，成为一名优秀运维工程师还需要

很多其他方面的能力，比如：

- 强烈的学习欲望和钻研精神。
- 良好的沟通和团队协作能力。
- 技术分享、沉淀、总结、传承能力。
- 行业洞察力。
- 好的身体素质和吃苦耐劳精神。
- 良好的编程能力。
- 胆大心细，敢于承担责任。
- 规划能力。
- 好的心态和情商。

1.1.3　自动化运维工具

上面我们对比了一下优秀运维工程师和普通运维工程师的区别。读到这里想必大家心中也有了基本的了解。伴随着互联网的云时代和大数据时代到来，早年的运维工程师只需要管理几十台、几百台服务器，而到目前为止很多运维工程师需要管理几万台服务器。如果没有一套完善的运维工具，这样的工作量和维护成本是不可想象的。所以想要作为一个优秀的运维工程师，就要借助运维工具来让我们更好地完成日常运维工作。在这里笔者重点向大家推荐3款功能强大的自动化运维工具——Cfengine、Chef和Puppet，它们也被誉为自动化运维工具中的“三把斧”。下一节我们将向大家简单介绍一下这3款工具。

1.2　自动化运维工具箱

1.2.1　Cfengine

Cfengine是一个借助C语言开发的、功能强大的自动化UNIX管理工具，最早出现于1993年。通过Cfengine可以轻而易举地管理客户端上的设备。Cfengine不仅运行成本低、效率高、功能强大，而且使用范围广。Cfengine可以管理各种环境下的设备，从一台到上千台服务器的集群均适用。如果运维工程师想同时修改2000台服务器的root密码，通过Cfengine可以轻松地在几分钟内实现。Cfengine还包含以下主要的功能：

- 检查和配置网络接口。
- 编辑系统和用户的文本文件。
- 维护符号链接。
- 检查和设置文件的权限。
- 删除垃圾文件。

- 检查重要文件和文件系统的存在。
- 控制用户脚本和 shell 命令的执行。
- 基于类的判定结构。
- 程序和系统进程管理。

关于 Cfengine 更多信息，请大家参考 Cfengine 官方网站 http://cfengine.com/。

1. Cfengine 生命周期

Cfengine 工作时遵循的是系统生命周期管理的 Build-Deploy-Manage-Audit（BDMA）模式（如图 1-2 所示）。BDMA 包含系统生命周期的 4 个阶段：构建（Build）、部署（Deploy）、管理（Manage）和审计（Audit）。

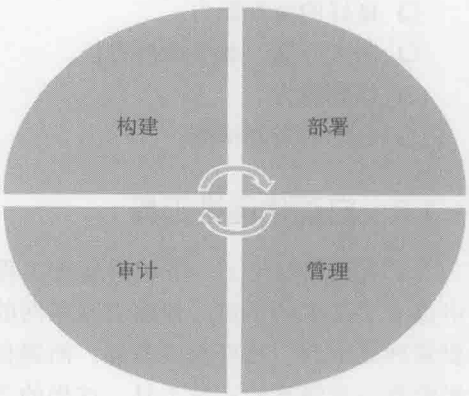


图 1-2 Cfengine 系统生命周期

下面分别介绍一下 BDMA 系统生命周期的 4 个阶段。

- 构建阶段：需要计划策略更改、规划想要的状态承诺（promise）以及构建所建议承诺的模板，这样如果所有机器均能做出并兑现这些承诺，系统便可无缝地运行。
- 部署阶段：需要向所有自主客户端（autonomous clients）发布策略，并且每个客户机都要运行一个代理，无需协助即可实现并维护这些策略。
- 管理阶段：这个自主代理负责管理该系统，运维工程师只需处理不能被自动处理的极少事件。
- 审计阶段：对更改进行本地审计和维护。决策结果由 Cfengine 内的设计确保且可自动维护。

2. Cfengine 如何工作

Cfengine 的工作过程如下。

1）管理员登录主服务器更新配置文件（SVN），通过运行 Cfrun 命令通知客户端进行更新。Cfrun 在 Cfrun.hosts 文件中查找客户端的列表。

2）Cfrun 与每个客户端上的 Cfservd 进行通信，然后 Cfservd 运行 Cfagent。

3）Cfagent 连接主服务器，首先检查 Update.conf 是否有新版本，如果有更新，则将它传输到客户端。

4）Cfagent 先评估 Update.conf 的内容，并获取策略文件（Cfagent.conf 和相关文件）的最新版本。随后评估 Cfagent.conf 以确定客户端是否处于所需状态。如果有偏差，Cfagent 将执行已定义的操作来更正客户端配置。

1.2.2　Chef

1. 什么是 Chef

自动化管理工具 Chef[⊖]由 Ruby 语言开发，是一种可以将框架转换为代码的自动化工具平台。人们不必关心设备是虚拟机还是物理机，也不必关心是几百台还是几千台服务器的集群，Chef 都可以方便自如地管理资源、进程、系统等信息。目前很多互联网公司也在应用 Chef，如 Facebook、亚马逊等。

Chef 服务器能够存储用户的配置数据和 Recipes，其中配置数据用于描述基础设施的所有组成部分，而 Recipes 将这些组成部分集合为一个完整的运行系统的指南。无论是在现场的实体和虚拟服务器上，还是在云端的实体和虚拟服务器上，Chef 用户都可以在系统的节点使用 Recipes。随着基础设施的变化和发展，用户可以使用工作区域随时更新 Chef 服务器。通过使用版本控制可以获取所有的更改。

2. Chef 如何工作

如果忽略所有的细节，Chef 是这样工作的：在工作站（Workstation）上定义各个客户端（Client）应该如何配置，然后将这些信息上传到中心服务器，每个客户端连到中心服务器工作站查看如何配置，然后进行自我配置。因此，在 Chef 的环境搭建完成以后，绝大部分工作是在工作站上进行的，客户端要获取工作站配置时，会主动连接并按照工作站的配置应用到客户端上，客户端并没有额外的工作。Chef 主要有以下 3 种运行模式。

- Chef-Solo：由一台普通计算机控制所有的服务器，不需要专设一台 Chef-Server。
- Client-Server：所有的服务器作为 Chef-Client，统一由 Chef-Server 进行管理，包括安装、配置等工作。Chef-Server 可以自建，但安装的东西较多，由于使用 Solr 作为全文搜索引擎，因此还需要安装 Java。
- Opscode Platform：类似于 Client-Server，只是 Server 端不需要自建，而是采用官方网站提供的 Chef-Server 服务。

上面 3 种管理模式中，无疑 Client-Server 模式是最好的，但是同时也是最复杂的。因为通过这一模式可以在本地环境中搭建一个私有的 Chef 集中管理环境，而无需依赖任何第三方平台。

1.2.3　Puppet

1. 什么是 Puppet

Puppet[⊖]是一款使用 GPLV2X 协议授权的开源管理配置工具，用 Ruby 语言开发。其既

⊖ Chef 官方网站 http://www.getchef.com/。

⊖ Puppet 官方网站 http://puppetlabs.com/。

可以通过客户端－服务器的方式运行，也可以独立运行。Puppet 可以为系统管理员提供方便、快捷的系统自动化管理。对于系统管理员来说通过 Puppet 配置管理系统，底层的操作系统的发行版本是透明的，Puppet 通过（Provider 又称提供者）属性来完成软件的配置与安装，管理员不必关心操作系统的种类与发行版本，如图 1-3 所示。Puppet 还可以提供一个强大的框架来完成系统管理功能，在框架的基础上系统管理员可以通过 Puppet 语言来描述系统的一些事务，如安装软件、初始化系统、启动、删除服务、推送配置文件和差异化配置管理服务器等。同时系统管理员和系统管理员之间可以分享用 Puppet 语言描述好的事务，从而减少重复劳动，提高工作效率。

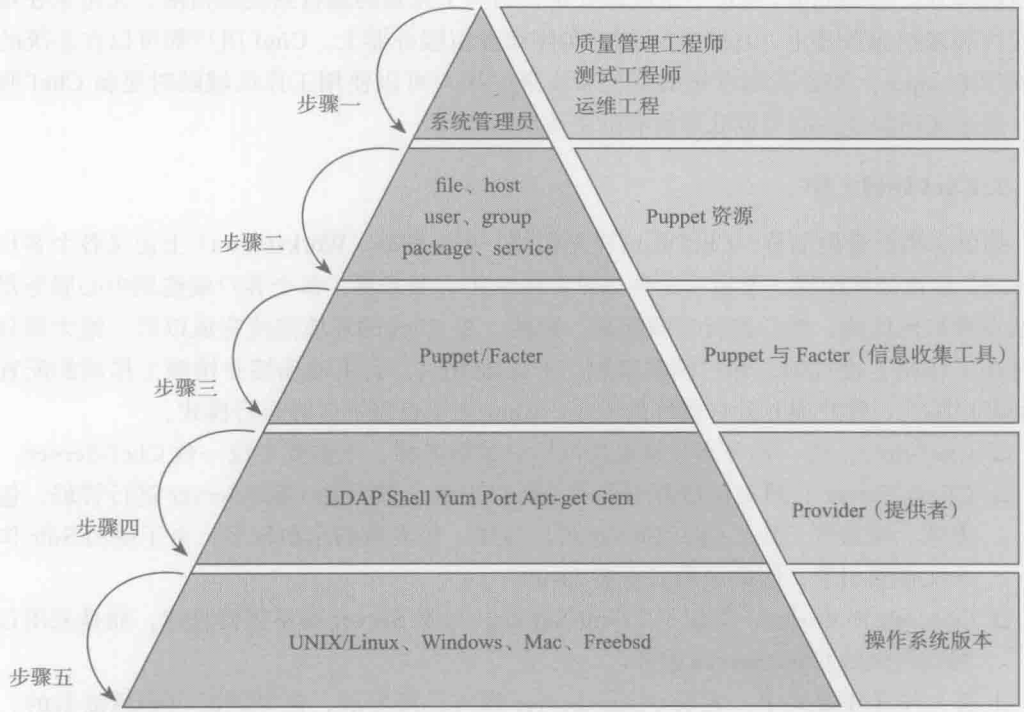


图 1-3 Puppet 管理操作系统流程图

Puppet 主要由 Luke Kanies 和他的公司 Puppet Labs 开发和维护。Kanies 从 1997 年开始从事 UNIX 的系统管理，并于 2005 年创立了一家专注于自动化工具的开源软件公司 Puppet Labs。不久之后，Puppet Labs 发布了他们的旗舰产品 Puppet。

2. Puppet 工作模型

Puppet 可以用来管理 UNIX/Linux 平台，同时也添加了对微软 Windows 的支持（要注意的是目前 Puppet 只对微软的 Windows 做客户端支持，并不能将 Windows 用作 Master 来管理其他的服务器）。

那么 Puppet 是如何工作的呢？目前 Puppet 有一个简单易懂的工作模型，如图 1-4 所示。其主要分为 3 层，分别是部署和调度层、配置语言和资源抽象层、事务层。

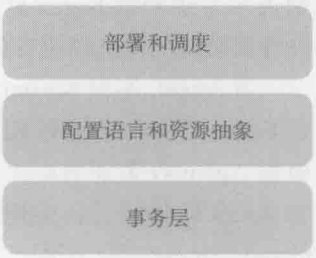


图 1-4　Puppet 工作模型

（1）部署和调度层

Puppet Master（Puppet 服务器，下称 Master）在一台机器上以守护进程的方式运行，同时还包含各客户端节点的配置信息。Puppet Agent（客户端，下称 Agent）在与 Master 通信的过程中，通过标准的 SSL 协议进行加密和验证，验证通过后，Agent 从 Master 上读取相应的节点配置信息。

需要注意的是，并不是每次连接 Agent 都会从 Master 上读取信息，只有该节点在 Master 上配置信息发生变化时才会被读取。

默认情况下 Agent 每 30 分钟连接一次 Master。但是这种方式在很多场景下不是很符合系统管理员的要求，所以很多系统管理员也会将 Agent 通过 Crontab（UNIX 定时任务计划）来管理，这样会更加灵活一些。

（2）配置语言及资源调度层

Puppet 使用描述性语言来定义配置项，在 Puppet 中将配置项被称为 Resource（资源）。这种描述性语言使得 Puppet 与其他配置工具截然不同。描述性语言在 Puppet 中还可以声明配置的状态，例如一个软件安装、配置、启动的各环节、上下游依赖关系等。

让我们来看这样一个例子。系统管理员需要在 CentOS、Ubuntu 的主机环境安装 Nginx 服务，如果不借助工具我们需要通过脚本来完成以下步骤。

步骤 1　连接到目的主机，输入用户密码或密钥。

步骤 2　检查是否安装相应 Nginx 服务。

步骤 3　如果没有安装，根据系统发行版本在系统上执行不同的命令。CentOS 可以执行 yum 命令，Ubuntu 可以使用 atp-get 命令，安装后启动 Nginx。

步骤 4　将安装后的信息返回给服务器。

而通过使用 Puppet，我们只需要在 Master 服务器的相应配置文件中通过配置语言定义一个 Package 资源。Package 资源的定义格式如下：

```
资源名 { '标题':
    属性 => 值
}
```

例如：

```
package { 'nginx':
  ensure => present,
}
```

在资源名里填写相应的标题，如 nginx，并将资源的属性 ensure 赋值为 present。这里

的属性 ensure 表示软件包的安装状态，present 表示希望安装这个 Nginx 软件包，而 absent 则表示希望卸载 Nginx 软件包。这样当 Agent 来连接 Master 时就会自动安装 Nginx 服务，并将安装好的消息以报告的形式上报 Master。

当 Agent 连接 Master 时，Master 并不知道 Agent 的操作系统型号和版本。Agent 通过 Facter 工具收集系统相关信息，并通过 SSL 协议将 Agent 的信息传递给 Master。Master 根据 Agent 收集到的相关信息，通过资源的提供者来为 Agent 服务。比如 Package 资源收到 Agent 的信息后，会识别 Agent 的系统型号版本，并通过资源提供者（如 yum aptitude pkgadd apt-get 等）匹配，为 Agent 服务。

（3）事务层

Puppet 事务层其实就是它的解析引擎。Puppet 事务层配置每一台主机的过程包括以下 4 步。

步骤 1 解析和配置编译。

步骤 2 将编译好的配置同步到 Agent。

步骤 3 在 Agent 上应用配置。

步骤 4 向 Master 报告运行结果。

首先 Puppet 会创建一个图表来表示所有资源的关系和上下游执行顺序，以及和 Agent 的关系。然后 Puppet 将按照资源之间的关系和上下游顺序依次执行。

接着 Puppet 为每一个 Agent 获取相应的资源，并把它们编译成“目录”，然后将目录依次分发到各主机，并通过 Agent 来应用它们，最后应用结果以报告形式反馈给 Master。

Puppet 并不是完全的事务，因为事务会记录日志，而 Puppet 并没有记录日志，也无法像数据库那样进行回滚。

1.3 自动化运维工具对比

在 1.2 节中我们介绍了现在比较常见的自动化配置工具 Cfengine、Chef 和 Puppet，下面再来看一下这 3 款自动化运维工具的区别，如表 1-1 所示。

表 1-1 自动化运维工具区别

	Cfengine	Chef	Puppet
开发语言	C 语言开发	Ruby 语言开发	Ruby 语言开发
语法	难理解	难理解	容易理解
使用人群	用户较多	用户较少	用户较多
学习门槛	门槛高	门槛高	门槛低
安装	复杂	复杂	简单
安装配置文件	较多	较多	较少

讲到这里，我们已经基本了解了 Cfengine、Chef 和 Puppet 这 3 款自动化运维工具。通

过表 1-1 可知，Puppet 的优势还是比较明显的。若是我们去 Puppet 的官方网站[⊖]上看一看，会发现很多使用 Puppet 作为公司自动化运维工具的例子，目前超过 18000 家公司在使用 Puppet 软件，其中包括 Twitter、Zynga、美国银行、纽约证券交易所、迪斯尼、Citrix、Oracle、Google、Adobe 和 Evernote 等。另外大型招聘信息搜索引擎 Indeed.com 的数据对各大公司招聘职位描述统计发现，很多公司增加了对 Puppet 技能的要求，如图 1-5 所示。

图 1-5　Puppet 技能职位工作趋势

若是大家一直在关注 Puppet 的动态，一定会发现这则令人振奋的消息："2014 年 6 月 Puppet Labs 宣布获得 E 轮融资 4000 万美元"，投资者包括思科、Google Ventures、KPCB 和 Vmware 等，这无疑为 Puppet 的未来发展注入了更多的活力。既然 Puppet 具有如此多的优点和明朗的发展前景，就让笔者通过本书带领大家一起走进 Puppet 的自动化运维的世界吧。

⊖ http://puppet labs.com/about/customers。

Chapter 2 第 2 章

Puppet 介绍

本章主要希望读者在掌握 Puppet 之前对它的版本情况、工作流程和常见问题有一些基本的了解。首先介绍当前流行的 DevOps 运动，它是一组过程、方法与系统的统称，用于促进开发、运维和质量保障（QA）部门之间的沟通、协作与整合，Puppet 也是它重要的工具成员之一。其次介绍 Puppet 发行版本状况，并重点介绍开源社区版本的细节，如何升级，以及它对各系统发行版本的支持情况。然后介绍 Puppet 的基本工作流程，让读者了解 Puppet 内部的工作逻辑，为后续深入学习作铺垫。接着介绍 Puppet 的代码开发工具，它可以帮助提升工作的效率，降低程序出错概率。最后还会解答 Puppet 官网社区网友的一些共同问题和疑惑。

2.1 DevOps 介绍

Puppet 是 DevOps 运动重要的工具成员之一，所以在介绍 Puppet 之前首先来了解一下 DevOps 运动。DevOps 是英文 Development 和 Operations 的组合，是一组过程、方法与系统的统称。DevOps 有助于促进开发、运维和质量等部门之间的沟通、协作与整合，如图 2-1 所示。

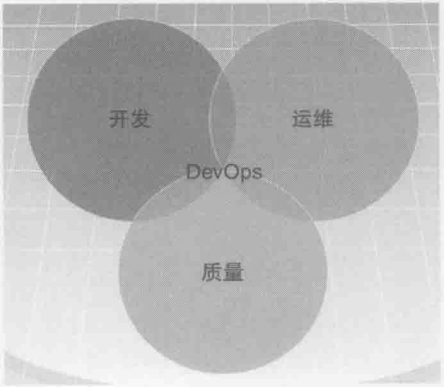


图 2-1 DevOps

在传统的软件组织中，将开发、运维和质量等设为各自独立的部门，这样不仅降低了各部门之间沟通的效率，同时也会引发很多问题。如，开发部

门接到项目经理和产品经理的指示需要开发一款新产品，而目前市场上还没有类似的产品。如果公司能够在短时间内开发出此产品，则不仅能填补相应的市场空白，也可以为公司带来可观的收益。而实际情况往往是这样的：各部门之间没能有效地沟通便开始了各自的工作，很可能开发部门在未与运维部门做足够的沟通前，便没日没夜地开发起这款新产品，最后终于在限期内将这款新产品开发出来，然后将开发后的相关产品一次性推给运维上线和后期维护。而当开发将新产品交于运维部门后问题便产生了，运维部门发现目前线上系统从硬件到软件都比较老，不适合这款新产品运行，无奈之下运维部门只能没日没夜地加班更新线上软硬件系统，强行将新产品运行在线上系统。这不仅容易导致新产品线上故障频出，给公司带来经济损失，同时也伤害了用户的产品体验，最后新产品以失败告终。在实际工作中这些问题层出不穷，但并不是无计可施。DevOps 运动的出现就是为了解决软件行业存在的这些问题，而 Puppet 就是 DevOps 运动中一个重要的工具成员，作为集中管理配置工作同时面向开发与运维。正如 Puppet Labs 的运维总监 Kartar 所说：DevOps 运动就是试图避免重大失误，并更聪明且高效地工作，它是一种旨在促进开发和运维两个团队相互合作、学习的思想、原则和框架。在一个 DevOps 环境中，为开发人员和系统管理员建立关系、流程和工具，让他们可以更好的交互，并最终为产品提供更好的服务。

2.2 Puppet 版本介绍

Puppet 目前提供两种发行版本，即开源社区版本和企业版本，我们稍后会介绍两个版本的区别。而本书从应用的角度出发，主要介绍开源社区版本，对企业版只做了解性的介绍。截至本书出版前，Puppet 官方网站共为用户提供了 4 个 Puppet 开源社区版的版本分支，不同版本分支之间主要是性能和功能上的差别（注：本书中介绍的案例多以开源社区 2.7.25 版本为例来做介绍）。目前 Puppet 支持不同分支版本之间的混用，稍后我们会介绍如何混用。但是为了避免不必要的麻烦，笔者建议尽量不要混用版本。下面来了解一下 Puppet 版本号介绍、版本混用可行性、如何升级 Puppet 和 Puppet 发行版本的具体情况。

2.2.1 Puppet 开源社区版本号介绍

目前开源社区版提供的 4 个版本分支如下：从版本 0.22.1 到 0.25.5 是 Puppet 的早期分支，目前官网已经不再提供技术支持；从版本 2.6.0 到 2.6.18 是目前正在维护的安全分支，已经停止开发；从版本 2.7.0 到 2.7.26 是目前正在维护的安全版本分支，也是目前比较主流的版本分支；从版本 3.0.0 到 3.6.* 是当前开发中的版本分支。

这里笔者推荐使用 Puppet 2.7.25 或 3.6.2 的 Puppet 版本。

细心的读者可能会发现，从第一类分支到第二类分支，Puppet 的版本从 0.25.5 版本一下跳到了 2.6.0。为什么 Puppet 版本号会跳了这么多呢？是 Puppet 2.6.0 版本比 0.25.5 版本

强大了 11 倍吗？答案是否定的。从官网上我们了解到，2.6.0 版本包含大量的附加特性，并且移除了 XML-RPC 传输层，改用 REST API，这样大大提高了 Puppet 的性能。读者需要了解的是 Puppet 版本号跳跃大，并不代表 Puppet 的发展速度和变化也很大，而只是代表其稳定性和功能性有所增加和改善。关于 Puppet 的版本，更多信息请参考官方网站 http://docs.puppetlabs.com/release_notes/index.html。

2.2.2 Puppet 版本混用可行性

最常见的 Puppet 部署模型是 C/S（客户端 / 服务端模型）。读者可能会问：是否能使用不同的 Puppet 版本作为 Master 和 Agent？答案是肯定的，但前提是要遵守如下的注意事项：

- Master 的版本一定要高于 Agent。例如，你可以将一个 0.24.8 版本的 Agent 连接到一个 2.6.0 版本的 Master，但是反过来是行不通的。
- Agent 的版本越老，在与新版本的 Master 搭配时正确运行的可能性就越小。一个 0.20.0 版本的 Agent 搭配一个 2.6.0 的 Master 基本上不可能正确运行。通常 0.24.x 版的 Agent 可以正常连接到 2.6.x 和 0.25.x 版本的 Master 并且正常运行。而更新版本的 Master 就可能无法完全兼容早期的 Agent，在使用时一些功能和特性可能会出现异常。
- 将 2.6.x 或更新版本的 Master 与 0.24.x 及早期版本的 Agent 混合使用意味着我们将无法获得 2.6.x 版本提供的全部性能提升。0.24.x 版的 Agent 依然会使用较慢的 XML-RPC 传输层来进行通信，从而无法利用新的 REST 接口。更多信息请参考官方网站 http://docs.puppetlabs.com/references/index.html。

2.2.3 如何升级 Puppet

在正式介绍升级版本的操作步骤之前，先来看一看 Puppet 升级版本中的注意事项。

1. 注意事项

目前 Puppet 官方网站对 Puppet 的版本还在不断更新中，如果目前使用的版本比较老，已经不能满足我们的需求，就需要通过升级新版本来支持更多的功能。在升级新版本时建议注意以下 3 点：

- 升级前仔细阅读 release note（http://docs.puppetlabs.com/release_notes/index.html），了解版本与版本之前的差异，以避免版本之间的差异带来的升级风险。
- 通常使用的 Puppet C/S（即 Client 为 Agent，Server 为 Master）架构来管理服务器，所以升级 Puppet 时，首先要要升级 Master，确保 Master 为最高的版本，再升级 Agent。
- 升级 Puppet 时尽量避免跳大的版本。如将现在用的是 2.6.* 版本直接升级到 3.6.* 版本，这样是不推荐的。建议先升级到 2.7 版本，待稳定没有问题后再平滑地过渡到

3.6.* 版本甚至更高的版本。

2. 升级步骤

升级的 Master 如果是一组服务器来提供服务的，那么建议先灰度[⊖]升级其中的一台服务器，并观察升级后的 Master 日志状态，通过日志的最终状态来评估是否升级全部的服务器。如果升级的 Master 是一台服务器，推荐以下的升级流程。

1）以源码方式安装 Puppet。安装新版本的目录与老版本目录分开。

2）在 Master 上以升级后的新版本 Puppet 来启动服务端口。这里可以通过 masterport 参数启动自定义端口，如 puppet master --no-daemonize --verbose --masterport 8141 --pidfile=/tmp/8141.pid 方式在 Master 启动 8141 端口与旧版本 Puppet 的 8140 端口来共同提供服务。在 Master 启动升级后的 Puppet 通过 Agent 来测试最终升级的结果，这里 Agent 需要通过“：端口”方式指定 Master 的新版本自定义服务端口，如 puppet agent --server puppet.example.com: 8141 --environment=production --test 方式，来最终确认是否能连接成功，并获取相应的配置。

3）观察 Master 的日志，确定新版本升级没有问题。如果需要还可以删除老版本目录，建立软链接到新版本目录。

4）整个升级结束。

2.2.4 Puppet 发行版本介绍

Puppet 的官方网站提供两种版本，即企业版本和开源社区版本。本书主要以开源社区版本为例进行介绍，因为企业版本大于 10 个节点是收费的，所以在这里只进行了解性的介绍。企业版与开源社区版本提供的服务器管理配置功能基本一致，但是企业版本会提供丰富的扩展功能，如表 2-1 所示（N 代表不支持，Y 代表支持）。

关于企业版本这里值得一提的是亚马逊 EC2（Elastic Compute Cloud，中文翻译即“亚马逊弹性计算云”）和 Vmware vms 功能，其中亚马逊 EC2 的用户可以支付一定的费用，在亚马逊购买云主机为互联网用户提供自己的服务。云主机的好处是可以降低成本，如果希望还能降低云主机运维成本，推荐读者使用企业版本的亚马逊 EC2 功能来为你管理云主机。另外就是 Puppet 的 Vmware vms 功能。了解 Vmware 的朋友应该都知道它是一家做虚拟化的公司。伴随互联网的浪潮到来，用户对互联网的要求越来越高，企业需要大量的服务器来支撑自己的服务，为了节约资源就需要用到虚拟化来合理分配资源。一般 Vmware 公司的虚拟化产品是首选，而 Puppet 企业版本的 Vmware vms 功能就可以支持这一虚拟化的管理，为我们进一步降低运维成本。目前企业版本 10 个节点以下是完全免费使用的，大于 10

⊖ 灰度是指在黑与白之间能够平滑过渡的一种发布方式。AB test 就是一种灰度发布方式，让一部分用户继续用 A，一部分用户开始用 B，如果用户对 B 没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到 B 上面来。灰度发布的优势是灰度期间就可以发现与调整出现的问题，当发布出现的问题比较严重时，可以第一时间回退操作，将损失降到最低程度。

个节点则需要向 Puppet 官方支付一定费用才能使用。

表 2-1 Puppet 企业版和开源版本区别

特 征	开源社区版本	企业版本
图形化管理	N	Y
亚马逊 EC2	Y	Y
Vmware vms	N	Y
management – Discovery	N	Y
management – User accounts	N	Y
Operating systems & applications	Y	Y
Puppet Forge 站点 1600+ 模块支持	Y	Y
官方 7×24 技术支持	N	Y
权限控制管理	N	Y
Puppet Lab 工程师认证	N	Y

关于企业版本和开源社区版本更多信息请参考官方网站 http://puppetlabs.com/puppet/enterprise-vs-open-source/。

2.3 Puppet 版本运行环境和硬件要求

目前 Puppet 支持 UNIX/Linux 和微软 Windows 系列的操作系统。读者需要注意的是，Puppet 在 2.6.0 版本之后才支持微软 Windows 系列操作系统，并且只支持 file 资源符。综合来看，Puppet 的接入门槛还是比较低的，支持多个操作系统和多种发行版本，同时对硬件要求也是不高。下面我们来了解一下 Puppet 版本运营环境和硬件要求。

2.3.1 Puppet 版本运行环境

1. Linux 发行版

由于 Linux 系统本身的版本就很多，现在市场上主流的就有十几种之多，所以就导致了 Puppet 支持 Linux 的版本也很多。下面做简单的列举，以便读者根据自己所用 Linux 版本选择相应的 Puppet 版本。

- RedHat Enterprise Linux 版本 4 或更高版本
- CentOS 版本 4 或更高版本
- Scientific Linux 版本 4 或更高版本
- Oracle Linux 版本 4 或更高版本
- Debian 版本 5 或更高版本
- Ubuntu 版本 8.04 LTS 或更高版本
- Fedora 版本 15 或更高版本

- SUSE Linux Enterprise Server 版本 11 或更高版本
- Mandriva Corporate Server 4
- ArchLinux

2. BSD/UNIX/Other

市场上主流的 BSD/UNIX/Other 相关发行版本也是比较多的，以下是常见的发行版本支持状况。

- FreeBSD 版本 4.7 或之后的版本
- OpenBSD 版本 4.1 或之后的版本
- Other UNIX
- Mac OS X，版本 10.4（Tiger）或更高版本
- Oracle Solaris，版本 10 或更高版本
- AIX，版本 5.3 或更高版本
- HP-UX

3. 微软操作系统 Windows

目前 Puppet 只支持微软近年发行的操作系统，详细如下：

- Windows Server 2003 和 2008（Puppet 2.7.6 或更高版本）
- Windows 7（Puppet 2.7.6 或更高版本）

2.3.2 Puppet 硬件要求

Puppet 对硬件的要求并不高，以下是 Puppet 的一个基本硬件配置要求和支持管理节点服务器的状况。

- 最小配置是双核 CPU，1GB 内存。
- 推荐配置 2 ～ 4 核 CPU，4GB 以上内存配置，这样的配置大约可以管理 1000 个节点服务器。

笔者觉得这个配置基本可以满足日常小规模服务器的管理，不过还要看我们所在网络的状况和管理的内容。在跨网访问环境或推送比较大的数据文件都会导致 Master 的超时，从而影响正常使用，这就需要通过增加硬件配置、改善网络环境或配置 Puppet 集群来解决。关于这些问题的解决方案会在第 11 章详细讨论。

2.4 Puppet 工作流程

Puppet 既可以单机运行，也可以通过 C/S 的方式运行。不过大多数场景下还是用 C/S 方式来运行 Puppet。如图 2-2 所示为 C/S 场景下 Puppet 的工作流程图。

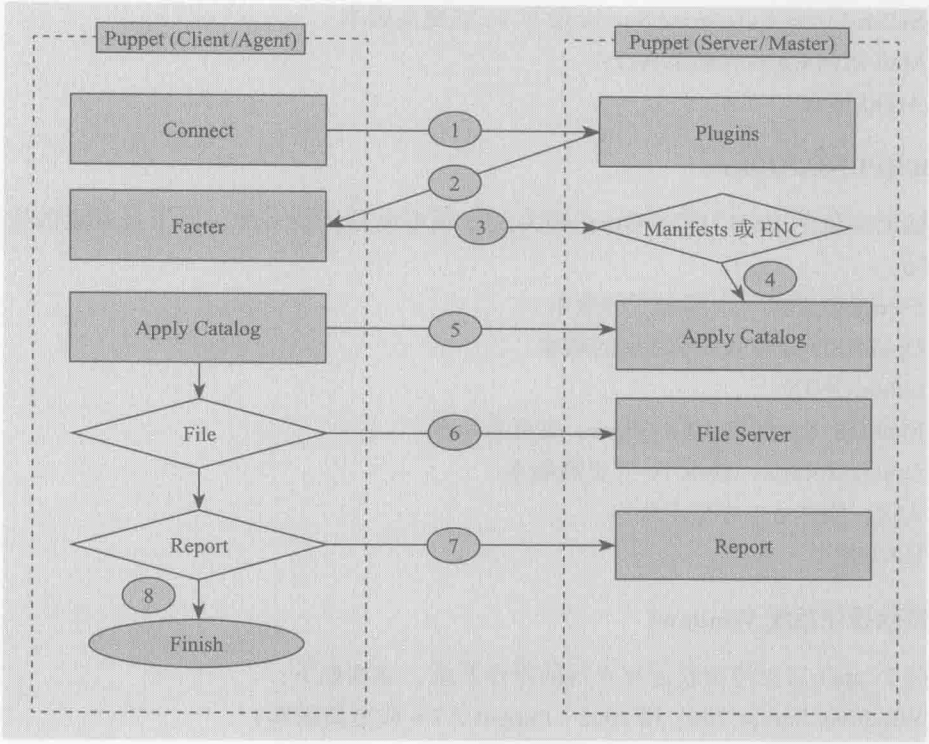


图 2-2 Puppet 工作流程图

下面具体分析 Puppet 的工作流程。

1）Agent 访问 Master 建立访问信任关系，流程中包括 Master 对 Agent 证书授权签名，并准许 Agent 访问 Master 资源。

2）建立信任关系后 Master 调用 Agent 的 Facter，探测出 Agent 主机的一些机器变量，如主机名、内存大小、IP 地址、CPU 和负载等信息。Agent 将这些信息通过 SSL 加密传输发送到 Master，Master 以变量名形式获取这些信息并使用。

3）Master 接收 Agent 的主机信息请求，并将它们发送到本机的 manifests 或 ENC（外部节点分类器），ENC 包括 Python、Ruby 和 Shell 等，只要支持 yaml 格式都可以，然后进行配置信息查询。

4）根据 Agent 的 HOSTNAME 匹配到相应的 Node（节点），整个解析过程分为几个阶段，首先是语法检查，如果有语法错误则停止报错，否则继续解析最终编译生成 Catalog。

5）Agent 接收到 Catalog 后，在本机应用 Master 的配置信息。

6）根据接收到的 Catalog 中的信息判断 Agent 在执行时有没有 File 文件要从 Master 推送到 Agent，如果有则向 Master Fileserver 发起请求获取文件。

7）将 Agent 的信息以报告形式上报 Master。读者需要注意 Puppet 2.6 和 Puppet 2.6 以

下版本默认并不会主动推送报告到 Master，需要设置 puppet.conf 配置文件中的 reports=true 参数。Puppet 2.7 和 Puppet 2.7 以上版本默认开启此功能。

8）流程结束。

2.5 Puppet 开发工具

工欲善其事，必先利其器。Puppet 官方不但提供了对服务器的配置管理解决方案，还为用户提供了多种开发工具以提高配置管理的效率。这里主要介绍 Geppetto 与 Vim 两种常用开发工具。关于 Puppet 更多的开发工具可以参考 http://projects.puppetlabs.com/projects/1/wiki/Editor_Tips。

2.5.1 Geppetto 开发环境

1. Geppetto 下载

Geppetto 是一款官方推荐的图形界面开发工具，可以帮助我们开发 Puppet 的 modules 和 manifests。Geppetto 通过 Eclipse 工具构建开发环境，它提供了语法高亮、内容补全、错误跟踪、代码调试和编译等功能。Geppetto 还通过接口在 Puppet Forge 上创建项目、编写自定义模块和上传自定义模块等。Geppetto 集成了 Git 与和 SVN 等功能，通过它们实现了对代码的版本控制、代码管理、语法分析以及版本比较等。下面来介绍 Geppetto 的下载与安装。

Geppetto 目前支持 3 种操作系统，它们分别是 Linux、MAC 与 Windows，如图 2-3 所示。Geppetto 支持 32 位的操作系统与 64 位操作系统。Geppetto 下载地址为 http://puppetlabs.github.io/geppetto/download.html。

图 2-3 Geppetto 官方下载网站

2. Geppetto 安装

目前 Geppetto 安装方式分为两种方式，一种方式是下载 Geppetto 后直接安装（Geppetto 需要 Java 环境支持，但是为了缩小 Geppetto 的安装包，目前 Geppetto 下载后并不提供 Java 环境支持，需要自行安装）；另一种方式是通过 Eclipse 来安装 Geppetto 扩展包。下面分别介绍。

（1）下载后直接安装

这里以 Mac 系统介绍第一种方式——安装 Java 和 Geppetto。

首先安装 Java，安装文件地址为 http://www.java.com/zh_CN/download/apple.jsp。安装前需要注意 Java 需要基于 Intel 的 Mac，该操作系统运行的是 Mac OS X 10.7.3（Lion）或更高版本，并且需要拥有管理员权限才能安装。在浏览器中下载 Java 的 .dmg 安装文件，下载后双击安装文件，如图 2-4 所示。

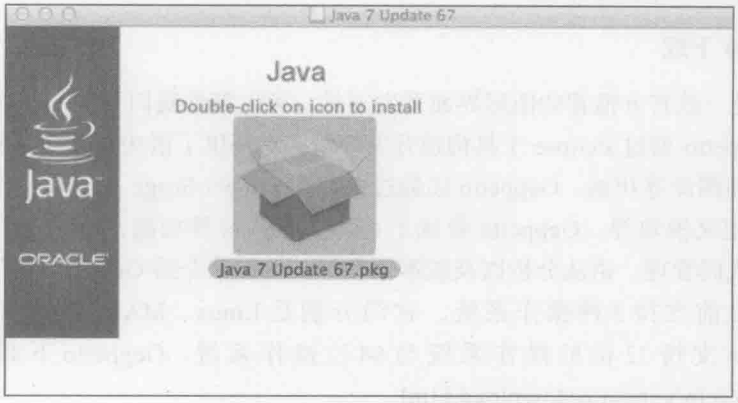


图 2-4 Java 源码文件包

安装 .dmg 文件会检查本机是否已经安装 Java，并根据检查结果进行完整安装 Java 源程序或升级 Java 源程序。由于笔者 Mac 系统已经安装了 Java，所以检查结果为对 Java 版本进行升级。

接着单击 Java 升级包名 Java 7 Update 67.pkg 进入安装环节，如图 2-5 所示。

安装环节中会提示我们 Java 的安装步骤与软件包在计算机上占用的磁盘空间，确认后单击“安装”按钮。

Java 成功安装后界面如图 2-6 所示。

接着下载 Geppetto 软件包，下载地址为 http://puppetlabs.github.io/geppetto/download.html。下载后为 .zip 格式文件，解压后进入目录后双击可执行文件 Geppetto。Mac 首次启动 Geppetto 可能会报“Geppetto 来自身份不明开发者”的信息，这里可以通过 Mac 系统的安全与隐私选项跳过此信息。打开 Geppetto 工具，如图 2-7 所示。这时就可以通过它开发 Puppet 的代码了。

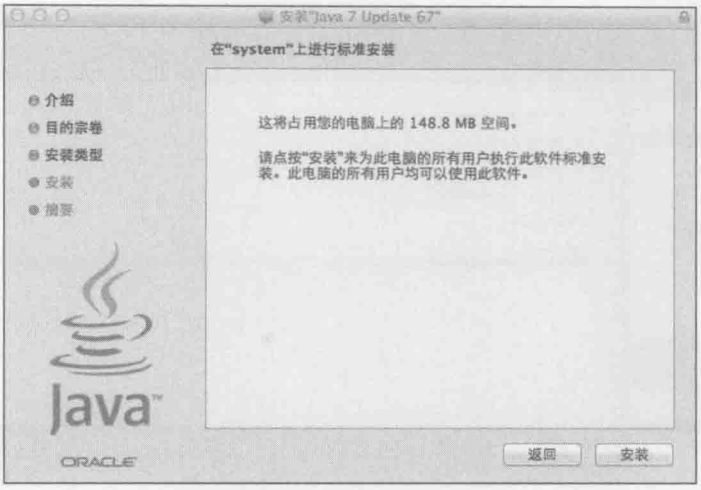


图 2-5 Java 安装环节

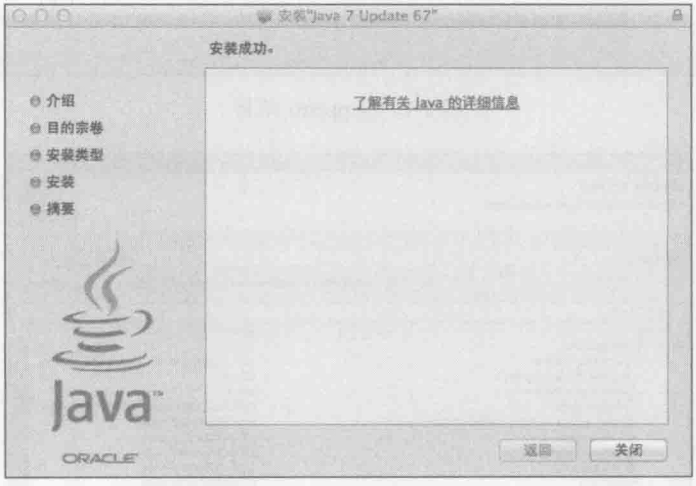


图 2-6 Java 7 成功安装界面

（2）通过 Eclipse 来安装

给 Eclipse 工具添加现有版本 Geppetto 的过程分为以下 5 步。

步骤 1 选择 Help -> Install New Software。

步骤 2 在 Work with 选项中增加 https://geppetto-updates.puppetlabs.com/4.x. 链接，如图 2-8 所示。

步骤 3 在 Eclipse 里选中 Geppetto，单击 next 按钮。

步骤 4 同意 license agreement，完成流程。

步骤 5 重启 Eclipse。

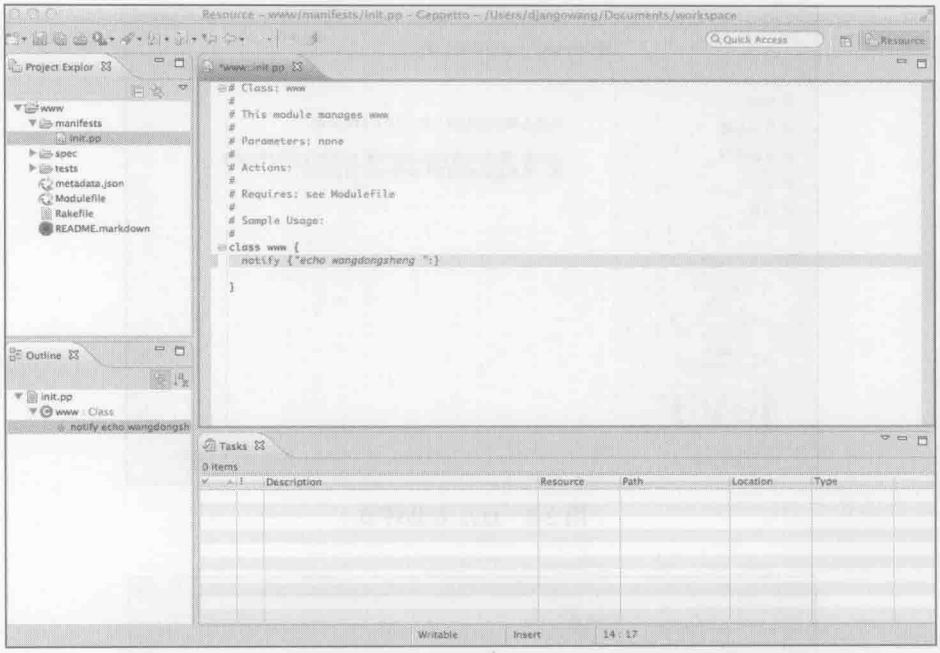


图 2-7 Geppetto 界面

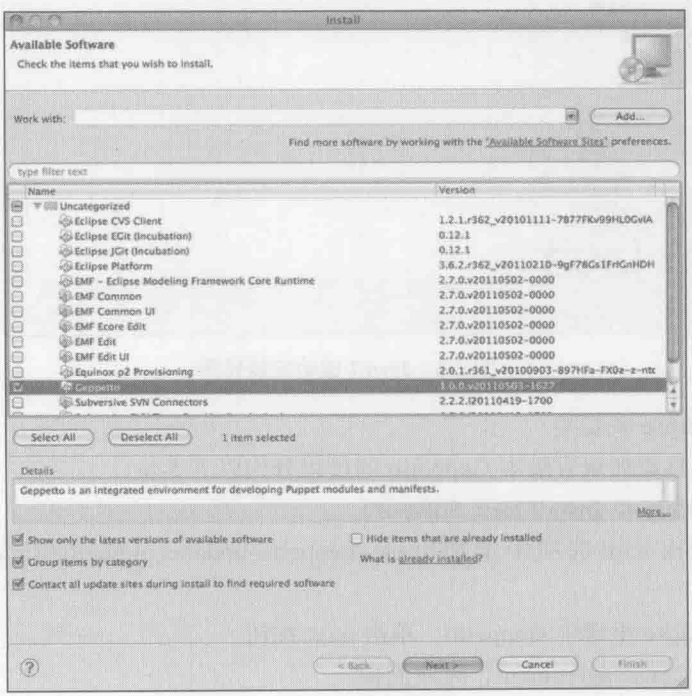


图 2-8 Eclipse 添加 Geppetto

2.5.2 Vim 开发环境

Vim 是由 Vi 发展出来的一个文本编辑器，它支持代码补全、语法高亮、插件扩展和编译及错误跳转等功能。使用 Vim 可以极大地提高编程的效率。Vim 在 UNIX/Linux 系列的程序员中使用广泛。目前 Puppet 的官方网站提供了 puppet.vim 的语法代码插件，语法代码插件功能主要基于 Puppet 的编程风格指南。它实现了语法高亮与自动补全功能，优势是不需要臃肿的 JRE 环境、启动速度快。下面我们通过 Puppet 官方网站来下载 puppet.vim 语法高亮的扩展文件。

```
# wget http://downloads.puppetlabs.com/puppet/puppet.vim -P /usr/share/vim/vim64/plugin
```

以上命令的作用是将 Puppet 官方网站的 puppet.vim 文件下载到 /usr/share/vim/vim64/plugin 目录中。-p 参数指定下载目录的位置，指定它的位置为 Vim 的扩展目录。下载成功确认没有问题后，再次编辑 *.pp 文件，会发现 Puppet 的代码文件内容已经高亮显示。

如果仍然无法高亮显示，可尝试以下步骤：

1）打开 vimrc 配置文件，添加 syntax on 打开语法高亮功能。

2）以 RedHat 为例编辑 /etc/profile 文件，追加 export TERM=xterm-color 到文件中，并执行 source /etc/profile 命令，重新加载配置变量。

2.6 Puppet 问答

在对 Puppet 的配置和功能有了初步的了解之后，很多读者可能不禁会对 Puppet 的开发语言、选用的语言格式等产生疑问，也会在对 Puppet 产生了兴趣之后有了自己的想法，想要参与到 Puppet 的开发中去。下面我们就对大多数读者都会关心的几个问题做出解答。

问题一 为什么选用 Ruby 语言开发 Puppet？

为什么选用 Ruby 作为 Puppet 的开发语言呢？ Puppet 的作者 Luke Kanies 是一名系统管理员，多数情况下 Luke Kanies 采用 Perl 来编写程序，但当他想要试着编写脑海里所想程序的原型时却不能从 Perl 中获得想要的类别关系。然后 Luke Kanies 尝试了 Python，因为这一想法产生于 2003 年，而那时写 Python 的人还比较少，几乎所有人都在谈论 Python 有多么的神奇，但是尝试之后 Luke Kanies 却觉得他完全不能用 Python 来编写想要的程序。一个偶然的机会，朋友向他提起 Ruby 的功能很强，所以 Luke Kanies 尝试了 Ruby，之后仅仅用了 4 个小时的时间，他就从完全没有接触过 Ruby 的新手变成了一个工作模型的熟手。从那以后 Luke Kanies 从未试着用其他语言，也从未后悔当初选择 Ruby 开发 Puppet。

在笔者看来，不同编程语言有不同的优势，没有最好的编程语言只有最适合的编程语言。Ruby 简单、方便、快捷和面向对象开发等优势，使它为 Puppet 后续蓬勃发展奠定了良好的基础。

问题二 为什么 Puppet 用自己的语言格式而不用 XML 和 YAML？

Puppet 语言使用的 manifests 有着非常人性化的操作界面。而 XML 和 YAML 作为两个围绕计算机处理能力而设计的数据格式，其人性化操作界面功能非常差。尽管一些人比较适应识读和编写 XML 和 YAML 的代码，而实际上人们也习惯于浏览网页而不是直接阅读 HTML。同理可得，使用 XML 和 YAML 的数据格式将会限制操作界面陈述表达的能力，而这一进程将会区别对待 XML 配置。

而 Puppet 2.6.0 实际上就加入了使用 Ruby 作为输入格式的功能，使得 Puppet 可以完全用 Ruby 语言来编写。然而，我们需要谨慎应用这一功能，并且需要避免可能出现故障的情况。Ruby 的完整语法功能往往特别多，我们相信系统管理者应该能够在更高级别的系统中对他们的数据中心予以模型化。性能和功能维护的最佳平衡点往往就是采用 Puppet DSL 来编写显示程序。

问题三 Puppet 都适合于什么场合？

Puppet 能使任何想要降低维护计算机成本的机构从中受益。然而，因为投资回报率与多种因素相关，例如当前的成本开销、现有计算机之间的差异、停机成本等，所以，任何机构在决定是否应将投资倾斜于配置管理工具，关于 Puppet 的花费多少才合适的情形下，都可以先仔细了解一下自身的实际情况。如果企业确实面临以下问题，则完全可以使用 Puppet 作为解决方案。

- 服务器管理的成本很高。
- 花费大量资金在停机维护上。
- 拥有大量近乎相同的服务器。
- 在服务器配置方面需要的灵活性和灵敏性。

问题四 如何向 Puppet 贡献代码？

Puppet 作为开源软件，其之所以广受追捧不仅仅在于配置和应用的实用性和灵活性，更在于其能够不断更新完善并吸纳好的建议。而 Puppet 也欢迎广大计算机爱好者向 Puppet 贡献代码。在对 Puppet 的理解和运用到达了一定境界之后，可以通过以下途径向 Puppet 贡献代码。

首先，可以加入一到两个邮件列表，官方推荐 http://groups.google.com/group/puppet-dev/ 和 http://groups.google.com/group/puppet-users/；或者可以加入 irc.freenode.netIRC。

如果读者热衷于开源软件发展和推动 Puppet 的进步；还可以访问 http://projects.puppetlabs.com/projects/puppet/wiki/Development_Lifecycle，这里有关于 Puppet 的开发和提交方式等信息。

第 3 章 Chapter 3

Puppet 及相关工具的配置与安装

万事开头难，所以本章首先重点介绍安装 Puppet 环境所需要的软件依赖包和各发行版本系统的安装步骤和注意事项，为读者应用 Puppet 奠定基础；然后介绍 Puppet 的辅助工具之一，即版本控制工具的安装方式，Puppet 与版本控制工具的整合，实现了线上配置与 SVN 强一致性的功能，同时可以对线上文件进行版本控制，以便出现问题时及时回滚，为提供稳定服务保驾护航；还介绍流程版本控制工具的安装与对比，进一步探讨在使用 Puppet 时应该选用的版本控制工具的安装方式及其区别；最后了解 Puppet 的辅助工具 DNSmasq，它是一款轻量级 DNS 工具，为我们管理 Agent 提供了很多的方便，这里介绍它的安装与应用。

3.1 Puppet 各环境的安装

第 1 章介绍过 Puppet 和其他的几个自动化运维工具的异同，Puppet 的优势有很多，安装方便只是其中之一。本节介绍 Puppet 在各主要环境下的安装方式。相信读者学完本章以后会深深体会到 Puppet 的安装是多么简单。由于 Puppet 是用 Ruby 语言编写的，所以先从 Ruby 版本支持 Puppet 状况讲起；接着介绍包管理工具和源；最后再来介绍 Puppet 在各环境下的安装方式。

3.1.1 Ruby 不同版本对 Puppet 的支持状况

目前 Puppet 提供了对多系统、多环境的支持，但不论是 UNIX/Linux 系列操作系统，还是微软的 Windows 系列操作系统，在安装 Puppet 前都需要先安装 Ruby，因为 Puppet 配

置的管理系统主要是由 Ruby 语言开发的。那么什么是 Ruby？简而言之，Ruby 是一种跨平台、面向对象的动态类型编程语言。

1995 年 12 月，松本行弘（Yukihiro Masumoto）[⊖]混合了他喜欢的语言发布了一种具有函数式及指令程序特性的新语言，并以发布的月份——7 月的诞生石（红宝石）为名，将其命名为 Ruby。

在安装 Ruby 前，先来看一下 Puppet 官网提供的 Ruby 不同版本对 Puppet 版本的支持情况，如表 3-1 所示。

表 3-1 Ruby 版本对 Puppet 版本支持

Ruby 版本号	Puppet 2.6	Puppet 2.7	Puppet 3.x
1.8.5*	支持	支持	不支持
1.8.7	支持	支持	支持
1.9.3**	不支持	不支持	支持
1.9.2	不支持	不支持	不支持
1.9.1	不支持	不支持	不支持
1.9.0	不支持	不支持	不支持
1.8.6	不支持	不支持	不支持
1.8.1	不支持	不支持	不支持
2.1.x	不支持	不支持	支持 3.5（或更高版本）
2.0.x	不支持	不支持	支持 3.2（或更高版本）

从表 3-1 中我们了解到，虽然 Ruby 的版本很多，但是 Puppet 并不支持所有的 Ruby 版本，主流 Puppet 版本对 Ruby 1.8.7 支持要好一些，所以这里推荐大家使用 Ruby 1.8.7 版本，如果有特殊需求可以根据表中 Puppet 支持状况来选择相应的版本。关于 Puppet 支持的 Ruby 版本的更多信息请参考官方网站 http://docs.puppetlabs.com/guides/platforms.html#ruby-versions。

3.1.2 包管理系统和源

包管理系统是一个软件的安装工具，它解决了软件安装和软件之间的依赖问题。各 UNIX/Linux 系统环境都会有自己相对独立的包管理系统，如 Fedora 和 RedHat 系统的 yum（Yellow dog Updater Modified），就是一个包管理系统，通过 yum 可以从指定的“源”自动下载 RPM 包并且安装，自动处理依赖性关系，并且一次安装所有依赖的软件包，这就是包管理系统的主要用途与优势。

我们再来看一下“源”。源是一些包含 URL 地址的文件，URL 地址为安装软件安装包的位置。如果包管理系统是安装软件的工具，那源就是它的软件仓库，可以通过源与包管理系统的结合来满足安装不同软件的需求。下面来介绍系统“源”和 Puppet 源的安装。通常在使用系统源时会出现超时的情况，具体的解决方法我们将会在本节介绍。接着介绍“源”的搭建，很多企业内部网为了网络安全是禁止访问互联网的，使得软件包更新或安装需要借助其他工具实现。为了降低软件包管理的成本，我们可以搭建企业内部“源”来解决这个问题。

⊖ 松本行弘（MatsumotoYukihiro），日本人。中学二年级时在父亲的口袋型电脑 Sharp PC-1210 上以 Basic 写了第一个程式。1984 年进入筑波大学第三学群资讯（情报）学类。大学期间休学两年，从事基督教传教工作。大学时在程式语言研究室工作，1990 年毕业。1993 年以来，一直从事 Ruby 的设计与开发。

1.“源”和 Puppet 源的安装

以 RedHat 系统为例来介绍系统“源”和 Puppet 源的安装。首先来看系统源的安装，通常它们存放在 /etc/yum.d.report 目录下的文件中。我们在安装软件时经常出现超时的情况，这是由于这些源存放在国外的原因导致。为了避免类似问题的发生，提高源的稳定性，建议尽量使用国内的镜像源，目前像网易、搜狐和淘宝都提供了一些镜像源，它们会定时同步更新国外的源到国内的服务器，这样使得我们更新软件更加稳定。这里以网易提供的源为例介绍。将 CentOS6-Base-163.repo 文件下载到 /etc/yum.d.report 目录，其中 CentOS6-Base-163.repo 文件内包含了软件包的网络地址路径，下载方式如下：

```
# 网易镜像源
# wget http://mirrors.163.com/.help/CentOS6-Base-163.repo -P /etc/yum.report.d/
```

这样网易的系统源就安装好了，但是需要读者注意，源中的软件只包含发行版本中的常规软件，并不包含 Puppet 相关的软件，所以还要安装 Puppet 官网提供的源。Puppet 官方为了方便人们安装 Puppet，将源打包成了 RPM，所以可以通过以下方式来安装 Puppet 的源：

```
# 安装 Puppet 官方源
# rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
```

成功安装 Puppet 源后，再导入 Puppet 官网提供的 GPG 密钥（在网络传输中不免会发生丢包或者软件在互联网的转载过程中植入非官方提供的功能的情况，GPG 密钥的作用是验证包的完整性和与官网提供的原始包的一致性）。

```
# 导入 GPG 密钥
# rpm -import http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
```

最后系统会在 /etc/yum.d.report 目录下生成 Puppet 的镜像源。这时就可以用 yum 命令来安装或升级 Puppet 及其相关组件了，yum 命令如表 3-2 所示。PuppetLabs 官方网站还提供了不同系统环境的安装源，更多的内容可以参考 http://docs.puppetlabs.com/guides/puppetlabs_package_repositories.html。

表 3-2 yum 命令参数说明

Yum 参 数	说 明
-y	不需通过用户确认将要发生的操作
-y install < 软件名 >	安装指定的软件包
-y update < 软件名 >	升级软件包
check-update	检查是否有升级的软件
Info updates	显示所有升级软件包信息
List	显示所有已经安装的软件包
search < 关键字 >	查找指定关键字
remove < 软件包名 >	删除软件包名

2.“源”的搭建

源的搭建就是在企业内部网指定能访问公网的服务器，将某系统发行版本的源复制到该服务器上，所有内部网机器指定这台服务器为授权访问互联网的机器进而更新与安装软件。以笔者使用 CentOS 系统发行版本为例，分为 5 步来搭建：

1）选择国内镜像下载，这样下载速度会有保障。以下为搜狐提供的 CentOS 镜像下载

地址。

```
# http://mirrors.sohu.com/centos/6.5/isos/x86_64/CentOS-6.5-x86_64-bin-DVD1.iso
# http://mirrors.sohu.com/centos/6.5/isos/x86_64/CentOS-6.5-x86_64-bin-DVD2.iso
```

2）在本机创建临时挂载点，挂载下载后的镜像文件。

```
# mkdir ios_mount1 ios_mount2
# mount -o loop -t iso9660 CentOS-6.5-x86_64-bin-DVD1.iso ios_mount1
# mount -o loop -t iso9660 CentOS-6.5-x86_64-bin-DVD2.iso ios_mount2
```

3）复制挂载的镜像文件到指定目录，并创建 yum 索引。

```
# 创建挂载目录
# mkdir -p /data/yum_repodata/centos6.5/x86_64
# 复制镜像文件到挂载目录
# cp ios_mount1/*  /data/yum_repodata/centos6.5/x86_64
# cp ios_mount2/Packages/*  /data/yum_repodata/centos6.5/x86_64/Packages
# 创建 yum 索引
# createrepo -p -d -o /data/yum_repodata/centos6.5/x86_64
/data/yum_repodata/centos6.5/x86_64
```

注意 新加入的 RPM 软件包，需要通过 createrepo --update /data/yum_repodata/centos6.5/x86_64 更新本地源。

4）为本机的 yum 仓库搭建 Web 服务。以 Nginx 为例来搭建 Web 服务器。

```
# 安装 Nginx 软件包
# yum install nginx
# 修改 nginx.conf 文件，指定发布目录为刚创建的目录路径 /data/yum_repodata/
# root /data/yum_repodata/
# 启动 Nginx
# service nginx restart
```

5）在需要更新软件包的服务器上指定搭建好的源。

```
# 删除原有的新建 /etc/yum.repos.d/ 目录下后缀为 .repo 的文件
# rm -rf /etc/yum.repos.d/*.repo
# 新建 /etc/yum.repos.d/centos_6_5.repo，内容如下（注意：xx.xx.xx.xx 为自己搭建的 CentOS 6.5 源的 Nginx 监听的 IP，yyyy 为端口）
[base]
name=centos6
baseurl=http://xx.xx.xx.xx:yyyy/centos6.5/x86_64
gpgcheck=0
# 更新配置
# yum clean all
# yum makecache
```

3.1.3 在 RedHat 企业版或 CentOS 上安装 Puppet

在确认包管理系统和源已经安装的基础上再来安装 Puppet。Puppet 的安装顺序是首先安装 Ruby 和 Ruby-lib 相关扩展包，然后安装 Ruby-shadow（此包作用是在 Puppet 中可以通过 user 资源来管理系统账户的密码），最后安装 Puppet 的相关环境。

1. Ruby 的安装

在 RedHat 企业版或 CentOS 的发行版本上安装 Puppet 时推荐使用 yum，它可以根据计算出来的软件依赖关系进行相关的升级、安装、删除等操作。首先需要在系统中打开终端界面输入如下命令，来安装 Ruby 相关环境。

```
# yum install ruby ruby-libs ruby-shadow
```

2. Puppet 的安装

安装好 Ruby 环境后，以同样的方式可以在 Puppet 机器上安装 Puppet、Puppet-server 和 Facter。Puppet 软件包包含了 Agent 程序，Puppet-server 软件包包含了 Master 程序，Facter 是一个收集系统信息上报给 Master 的工具，具体的安装命令如下：

```
# yum install puppet puppet-server facter
```

3.1.4 在 Debian 和 Ubuntu 上安装 Puppet

Debian 和 Ubuntu 发行版本是个人用户使用比较多的版本。特别是 Ubuntu，笔者在初学 Linux 时也使用过 Ubuntu 发行版。Debian 和 Ubuntu 版本的安装顺序与 RedHat 企业版或 CentOS 的发行版安装顺序一致。具体步骤如下。

1. Ruby 的安装

在 Debian 和 Ubuntu 发行版本上安装软件推荐使用 apt-get。apt-get 是一个命令，适用于 Deb 包管理式的操作系统，主要用于自动从互联网的软件仓库中搜索、安装、升级、卸载软件。它和 yum 类似，可以帮助用户解决软件包的依赖问题。apt-get 以空格作为分割，安装 Ruby 与 libshadow-ruby 命令如下：

```
# apt-get install ruby libshadow-ruby
```

2. Puppet 的安装

安装好 Ruby 环境后，可以通过 apt-get 继续安装 Puppet。安装 Puppet 的 Master、Agent 和 Facter 的命令如下所示：

```
# apt-get install puppet puppetmaster facter
```

3.1.5 在微软 Windows 系列操作系统上安装 Puppet

在微软 Windows 系列操作系统上安装 Ruby、Puppet 和 Facter 也是非常方便的。官方提供了集成安装软件包供大家下载，下载地址为 https://downloads.puppetlabs.com/windows/。这里有一点需要特别注意，Puppet 对 Windows 支持的版本比较特殊，其从 2.6.0 版本后开始支持微软的 Windows 系列操作系统，目前支持的平台如下：

- Windows Server 2003/2003 R2
- Windows Server 2008/2008 R2
- Windows 7

目前 Puppet 只支持微软 Windows 系列系统的 Agent，尚未支持 Master。

要在 Windows 上安装 Puppet，只需到 Puppet 官网下载集成安装包后双击安装就可以了，如图 3-1 所示。集成软件安装包会自动安装 Puppet、Facter 和 Ruby 等相关软件，如图 3-2 所示。这部分内容非常简单，这里就不过多介绍了。

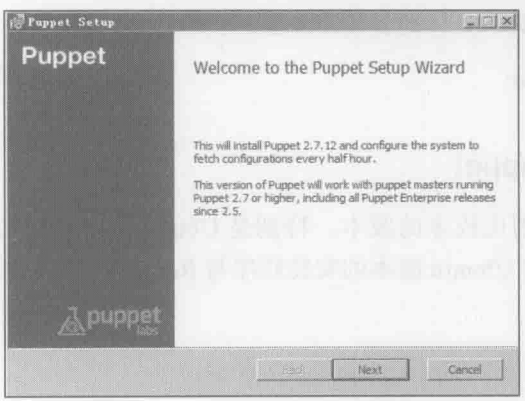


图 3-1 Windows 安装 Puppet

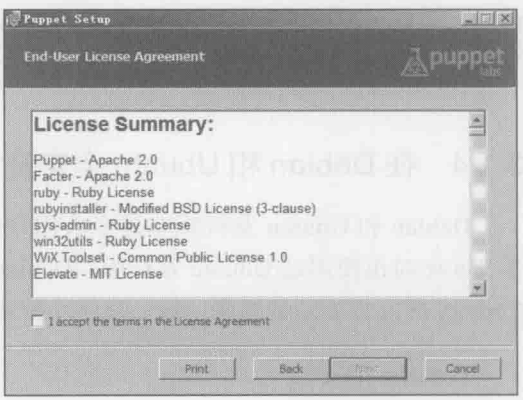


图 3-2 Windows 安装 Puppet 相关软件

3.1.6 在 Mac 上安装 Puppet

在 Mac（苹果系统）上可以使用 dmg 和 pkg 来安装软件，但是还可以通过更简单的 MacPorts 工具来安装。MacPorts 与刚介绍的 yum 安装工具非常相似，它可以帮助我们直接从网上查找软件，下载并安装它。首先安装 MacPorts 工具，然后再通过 MacPorts 工具来安装 Ruby，最后介绍通过官方网安装 Facter 和 Puppet 等工具。

1. 包系统安装

Mac 系统的包管理系统与其他发行版本有些不同，它有自己的独立安装方式。这里推

荐通过 MacPorts 工具来安装相关软件包。可以通过 http://www.macports.org/install.php 来下载 Mac 发行版本的 MacPorts 工具。

目前 Macport 支持 Mac 以下几种版本：

- OS X 10.9 Mavericks
- OS X 10.8 Mountain Lion
- OS X 10.7 Lion

更早版本，请参考 http://www.macports.org/install.php#installing。

```
# 这里笔者以 OS X 10.7 Lion 版本介绍，下载 MacPorts 工具
# wget https://distfiles.macports.org/MacPorts/MacPorts-2.2.1-10.7-Lion.pkg
```

下载后双击安装，如图 3-3 所示。安装后可以通过 port 命令来安装软件。关于 MacPorts 工具的使用，更多信息请参考官方网站 http://guide.macports.org。

图 3-3 MacPorts 安装界面

2. Ruby 的安装

在 Mac 系统上可以使用源码安装 Ruby，在这里推荐使用 MacPorts 工具来安装，这种安装方法比较简单，其功能与 yum 和 apt-get 类似。在 Mac 系统上打开终端，执行以下命令，它会自动从网上下载 Ruby-1.8.7 软件包并安装好。

```
# port install ruby ruby-libs
```

3. Facter 的安装

Puppet 官网提供 Mac 版本的 Facter 安装包，下载地址为 http://downloads.puppetlabs.

com/mac/facter-1.6.16.dmg。从网站上下载后直接双击软件包，按照指示顺序安装即可，如图 3-4 所示。

图 3-4 在 Mac 系统上安装 Facter

4. Puppet 的安装

Puppet 官网提供 Mac 版本的 Puppet 安装包，下载地址为 http://downloads.puppetlabs.com/mac/puppet-2.7.20.dmg。同样从官网下载后直接双击软件包，按照指示顺序安装即可，如图 3-5 所示。

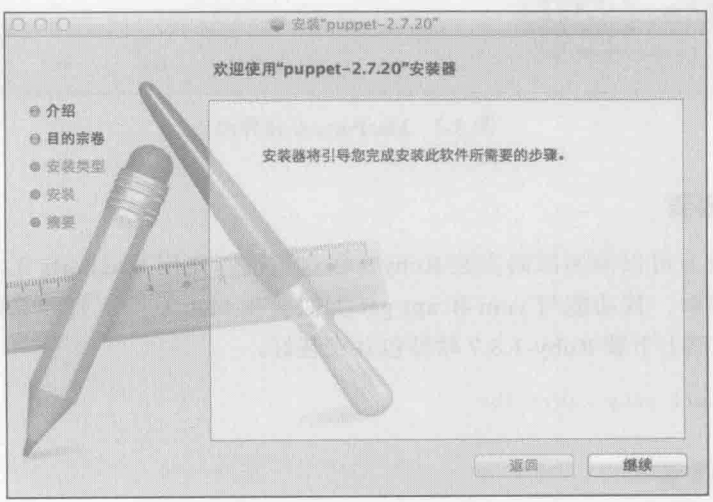


图 3-5 在 Mac 系统上安装 Puppet

3.1.7 通过 RubyGems 安装 Puppet

安装好 Ruby 后，也可以通过 Ruby 的 RubyGems 来安装 Puppet 和 Facter。RubyGems 是一个用于对 Rails 组件进行打包的 Ruby 打包系统，功能类似于 Linux 下的 apt-get 工具。它提供一个分发 Ruby 程序和库的标准格式，还提供一个管理程序包安装的工具。以下是通过 RubyGems 的 gem 命令来安装 Puppet 和 Facter 的方法。

gem 命令默认使用 Ruby 官方提供的镜像源，由于镜像源在国外不是很稳定，推荐使用淘宝提供的镜像源[⊖]，更换源方式如下：

```
#ruby.taobao.org
# gem sources --remove https://rubygems.org
# gem sources -a http://ruby.taobao.org
# gem sources -l
```

以下为通过 gem 命令来安装 Puppet 与 Facter。

```
# gem install puppet facter
```

RubyGems 会根据操作系统发行版本来自动匹配适合的 Puppet 和 Facter 版本。需要注意的是 Puppet 官网并不推荐以 RubyGems 的形式来安装 Puppet，因为 RubyGems 对比较老的 UNIX/Linux 发行版本中的 Puppet 支持并不好，通过它安装软件时可能会出现一些错误，特别是对于刚接触 Puppet 的用户，会增加学习的成本。

3.1.8 源码编译 Puppet

Puppet 源码编译安装与前几节介绍的通过系统（yum、apt-get、RubyGems）安装相比要稍微复杂一些，同样 Puppet 官方网站也不推荐以这样的形式安装，但是在一些特殊场景下，还是需要通过源码编译安装 Puppet。以笔者的工作环境为例，为确保网络安全，内网禁止访问互联网资源，笔者只能通过从互联网下载源码包，将源码包传入内网服务器，在内网一台服务器上通过源码编译的方式来安装 Ruby、Puppet 和 Facter。下面介绍通过源码方式安装 Ruby、Puppet 和 Facter 的过程。

1. Ruby 的安装

可以到 Ruby 官网（http://ftp.ruby-lang.org/pub/ruby/）下载 Ruby 1.8.7 版本。在安装 Ruby 时，Puppet 官方网站建议安装以下 Ruby 的扩展支持，它们通常是 Puppet 在配置管理服务器时需要用到的库与模块。

- base64
- cgi
- digest/md5

⊖ 关于淘宝 Ruby 镜像源，更多信息请参考官方网站 http://ruby.taobao.org/。

- etc
- fileutils
- ipaddr
- openssl
- strscan
- syslog
- uri
- webrick
- webrick/https
- xmlrpc

首先下载 ruby-1.8.7 的源码程序包，下载后解压，然后进入源码包。

```
# wget  http://ftp.ruby-lang.org/pub/ruby/ruby-1.8.7-p358.zip
# unzip ruby-1.8.7
# cd ruby-1.8.7
```

源码编译 Ruby 和很多传统的 UNIX/Linux 软件安装方法一样，分为 3 步，即 configure（配置）、make（编译）和 make install（安装）。安装过程如下。

步骤 1　配置 Ruby 环境，并通过 prefix 参数指定它的安装目录。

```
#  ./configure --prefix=/usr/local/puppet
```

步骤 2　编译 Ruby 环境。

```
#  make
```

步骤 3　安装编译好的 Ruby 环境。

```
#  make install
```

通过安装 configure 参数将 Ruby 安装到 /usr/local/puppet 目录下。Ruby 安装完成后，需要再次安装 Ruby 的扩展支持。这里需要注意一下环境变量，因为我们安装 Ruby 的位置并不在系统环境变量中，所以需要手动导入系统环境变量。安装过程如下：

```
#  export PATH=$PATH:/usr/local/puppet/bin/:/usr/local/puppet/sbin/
#  ruby  -r base64 -e "puts:installed"
#  ruby  -r cgi -e "puts:installed"
#  ruby  -r digest/md5 -e "puts:installed"
#  ruby  -r etc -e "puts:installed"
#  ruby  -r fileutils -e "puts:installed"
#  ruby  -r ipaddr -e "puts:installed"
#  ruby  -r openssl -e "puts:installed"
#  ruby  -r strscan -e "puts:installed"
#  ruby  -r syslog -e "puts:installed"
#  ruby  -r uri -e "puts:installed"
```

```
#  ruby  -r webrick -e "puts:installed"
#  ruby  -r webrick/https -e "puts:installed"
#  ruby  -r xmlrpc/client -e "puts:installed"
```

安装 Ruby-1.8.7 版本后还要安装 ruby-shadow（下载地址 https://github.com/apalmblad/ruby-shadow），之前已经介绍过它的用途，这里用的是 ruby-shadow-2.1.4 版本。

```
#  tar -xvzf apalmblad-ruby-shadow-2.1.4-0-g248703f
#  ruby extconf.rb
#  make
#  make install
```

2. Facter 安装

先要到 Puppet 官方网站 http://puppetlabs.com/downloads/facter/ 下载 Facter，下载后通过以下方式编译安装 Facter，这里以 facter-1.7.4 为例。

```
#  tar -zxvf facter-1.7.4.tar.gz
#  cd facter-1.7.4/
#  ruby install.rb
```

3. Puppet 安装

接下来安装 Puppet。Puppet 源码下载地址为 http://puppetlabs.com/downloads/puppet/。在下载 Puppet 后需要注意 Puppet 与 Ruby 对应版本，这里以 Puppet 2.7.25 为例。

```
#  tar -zxvf puppet-2.7.25.tar.gz
#  cd puppet-2.7.25/
#  ruby install.rb --full
```

通过 Puppet 源码编译安装后，切莫急着删除源码包，可以将源码包中的配置，如 puppet.conf、auth.conf、fileserver.conf 和 tagmail.conf 模板文件复制到 /etc/puppet 目录下。

3.1.9 源码打包 RPM

在前面讲解了通过源码编译的方式安装 Puppet 的原因，即一些企业内网为了网络安全禁止内网与互联网的服务器进行通信。利用上面介绍的方法，在一台或几台服务器上源码编译 Ruby 和 Puppet 还可以，如果机器比较多，那么进行源码编译也是需要一定成本的。所以推荐大家在一台服务器上安装好 Puppet 后，将源码打包成 RPM 的形式，然后通过 RPM 方式进行批量安装，这种方式比较方便快捷。

那什么是 RPM 呢？RPM 是 RedHat Package Manager（RedHat 软件包管理工具）的缩写，其名称虽然打上了 RedHat 的标签，但其原始设计理念是开放式的，包括 OpenLinux、

SUSE 以及 Turbo Linux 在内的很多发行版本都在使用。推荐在内网不连通互联网的情况下，使用 RPM 的安装方式，这样比较方便、快捷。关于 RPM 的打包方式可以在网上搜索到，这里不做过多的介绍。

3.2 版本控制工具安装与配置

Puppet 是一款配置管理工具，其优势不仅在于配置和管理线上系统文件，还可以利用版本控制工具对线上系统和配置文件进行版本控制，如图 3-6 所示。这样，当线上系统出现问题时可以根据版本控制工具进行及时回滚，而回滚的功能可以将配置文件回退到上一版本或历史的某一版本，快速的回滚可以让我们将故障时间降到最低。目前流行的版本控制工具有很多，在这里推荐两款版本控制工具——Subversion 和 Git，这两款工具各有优势，都可以与 Puppet 结合使用，本书中案例主要使用 Subversion 版本控制工具。以下讲解 Subversion 和 Git 的基本安装和两款配置工具的区别，以便读者选用适合自己的版本控制工具。

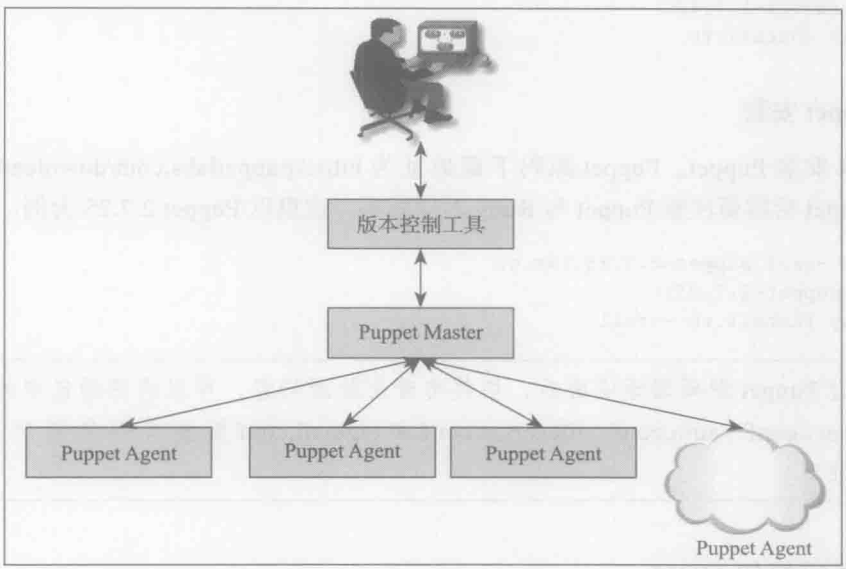


图 3-6 Puppet 与版本控制工具结合

3.2.1 Subversion 安装与配置

Subversion（简称 SVN）是近年来崛起的版本管理工具，也是 CVS 的“接班人”。目前绝大多数开源软件都使用 SVN 作为代码版本管理软件。SVN 服务器有两种运行方式，即独立服务器运行方式和借助 Apache 运行的方式。借助 Apache 运行方式更灵活一些，而且 Apache 有丰富的扩展功能，所以在这里介绍借助 Apache 方式运行 SVN，这种方式比较方

便快捷，容易上手。

1. Apache 下载与安装

首先需要下载 Apache（中文译名“阿帕奇”），它是一款流行的 Web Server，目前很多互联网网站均有使用。这里以 Apache 2.2.27（下载地址 http://mirrors.cnnic.cn/apache/httpd/httpd-2.2.27.tar.gz）为例，结合 SVN 介绍配置、编译和安装的过程。具体实现命令如下：

```
# tar xzvf httpd-2.2.27.tar.gz
# cd httpd-2.2.27
#"./configure" "--prefix=/usr/local/apache2" "--with-included-apr" "--enable-so"
"--enable-deflate=shared" "--enable-expires=shared" "--enable-rewrite=shared"
"--enable-static-support" "--disable-userdir""--enable-dav" "--enable-dav-fs"
# make && make install
```

2. SVN 安装

安装 SVN 前首先需要安装两个 SVN 的辅助工具——Sqlite 和 Neon，否则在使用 SVN 时会不支持一些功能。安装好 Sqlite 和 Neon 后再来安装 SVN。具体的安装命令如下：

```
#tar xzvf sqlite-autoconf-3071300.tar.gz
#cd sqlite-autoconf-3071300
#./configure && make && make install

#tar xzvf neon-0.29.6.tar.gz
#cd neon-0.29.6
#./configure && make && make install

#tar xzvf subversion-1.7.5.tar.gz
#cd subversion-1.7.5
#./configure --with-neon
#make && make install
```

3. Apache 配置

借助 Apache 可以管理 SVN 的 Httpd.conf 文件，所以需要对 Apache 进行简单配置。主要的配置信息如下。

1）加载 SVN 的模块。

```
# 添加如下模块支持
LoadModule dav_svn_module modules/mod_dav_svn.so
LoadModule authz_svn_module modules/mod_authz_svn.so
```

2）统一 SVN 和 Apache 的权限。

```
# 修改用户，保证访问权限
User puppet
```

```
Group puppet
```

3）设置 SVN 发布目录和权限。

```
# 设置 SVN 目录的访问
<Location /svn>
Order allow,deny
Allow from all
Dav svn
SvnParentPath /data1/svn
SvnListParentPath On
SvnAutoversioning On
AuthType Basic
AuthName "Subversion repository"
AuthUserFile /usr/local/apache2/conf/authfile
Require valid-user
</Location>
```

上述提到的 /usr/local/apache2/conf/authfile 包含了访问 SVN 的账户和密码信息，可以通过以下方式设置 SVN 管理账户的信息和密码。

```
#/usr/local/apache2/bin/htpasswd -c /usr/local/apache2/conf/authfile USRNAME
```

4. 启动 Apache

修改完配置文件后，不要忘记重启 Apache 后才会生效。启动命令如下：

```
# /usr/local/apache2/bin/apachectl restart
```

5. 确认安装

启动 Apache 后可以在 IE 浏览器输入 Apache 的 IP 地址，即配置文件中的 192.168.1.1。输入 IP 地址后，IE 会提示用户输入用户名和密码，如图 3-7 所示。这里的用户名就是 httpd.conf 配置文件中设置的认证文件地址（/usr/local/apache2/conf/authfile）中的用户名，密码是通过 htpasswd 命令设置的密码，如果登录成功表示已经成功地安装了 Apache 和 SVN。

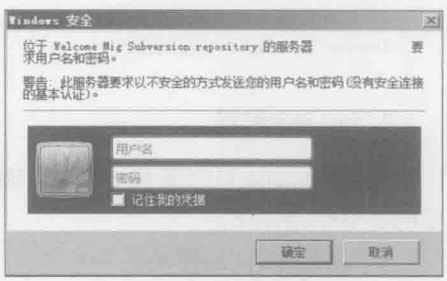


图 3-7 SVN 登录界面

3.2.2 Git 安装与配置

Git 是 Linus Torvalds（Linux 内核主要开发和创建人之一）为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。很多著名的软件都使用 Git 进行版本控制，其中包括 Linux 内核、X.Org 服务器和 OLPC 内核等项目的开发流程。Git 是强大的版本控制工具，与常用的版本控制工具 CVS、Subversion 等不同，它采用了分布式版本库的方式而不

需要服务器端软件支持。开发者从服务器上克隆数据库（包括代码和版本信息）到单机上，在没有网络的条件下，开发者就可以在单机上创建分支、修改代码，并在单机上进行提交，联网后可以与服务器上的版本进行合并。Git 就是如此方便快捷。更多信息请参考官方网站 http://git-scm.com/。

Git 目前支持 UNIX/Linux 系列操作系统，同时也支持 Windows 系列操作系统。下面以 CentOS 系统为例来讲解安装 Git 的过程。在安装 Git 前需要安装一些 Git 需要的依赖包，具体如下。

1. Git 依赖包

安装 Git 前首先通过 yum 安装它所需要的依赖包和库文件。具体安装命令如下：

```
# yum install curl
# yum install curl-devel
# yum install zlib-devel
# yum install openssl-devel
# yum install perl
# yum install cpio
# yum install expat-devel
# yum install gettext-devel
```

2. Git 安装

下载 Git（http://git-scm.com/download），确认依赖包和库文件安装完没有问题后，解压 Git 安装包直接配置、编译和安装它。具体命令如下：

```
# tar xzvf git-latest.tar.gz
# ./configure
# make
# make install
```

3.2.3 SVN 与 Git 的 4 点区别

SVN 和 Git 作为两款比较有代表性的版本控制工具，在功能和应用方面可谓各有千秋。两者的区别有以下几点。

1）Git 可以分布式管理版本库，而 SVN 不行。Git 和 SVN 一样都有自己的集中版本管理服务器。但是 Git 更倾向于分布式管理版本库，也就是说每个开发人员通过 Git 可以从中心服务器版本库上迁出（check out）一份代码到自己机器的版本库中，并且在没有网络的情况下仍然可以继续开发，并在本地提交相应数据。而 SVN 却做不到这一点。

2）Git 把内容按元数据方式存储，而 SVN 是按文件方式存储。在使用 SVN 和 Git 时会发现两个版本控制工具分别有 .svn 目录和 .git 目录。SVN 把文件的元信息隐藏在 .svn 文件夹里。而 Git 的目录体积要比 SVN 大很多，因为 Git 目录是处于机器上的一个克隆版的

版本库，它拥有中心版本库上所有的东西，例如标签、分支、版本记录等。

3）**Git 分支和 SVN 分支的不同**。SVN 中的分支一点也不特别，它就是版本库中另外的一个目录。如果我们想知道是否合并了一个分支，需要手工运行像 svn propget svn:mergeinfo 这样的命令来确认代码是否被合并。然而处理 Git 的分支却是相当的简单和有趣。我们可以从同一个工作目录下快速地在几个分支间切换。我们可以很容易发现未被合并的分支，而通过 Git 也可以简单而快捷地合并这些文件分支。

4）**Git 的内容完整性要优于 SVN**。因为 Git 的内容存储使用的是 SHA-1 哈希算法，能保证代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

在这里我们总结了 SVN 和 Git 的 4 点不同，读者可以根据自己的需要来选用 Git 或 SVN 版本控制工具。

3.3 DNS 安装与配置

DNS（Domain Name System，域名解析系统）主要用来表示 IP 与域名之间的映射关系，是学习 Puppet 过程中比较重要的一个辅助工具，因为在 Agent 每次连接 Master 时都会使用到域名。如果通过 Puppet 管理的 Agent 较少，可以通过写 hosts 的形式来管理域名，但是在管理的 Agent 比较多的情况下通过 hosts 的形式来管理域名的成本就相对较高。这时可以通过开源软件来搭建一套 DNS 域名解析系统，这样通过 Puppet 管理比较多的 Agent 就方便快捷多了。目前互联网比较常用的搭建 DNS 的软件是 Bind。其功能强大，但是配置复杂，所以在这里推荐一款轻量级的 DNS 软件——DNSmasq。

作为用于配置 DNS 和 DHCP 的工具，DNSmasq 小巧且方便，适用于小型网络，它提供了 DNS 功能和可选择的 DHCP 功能。它服务那些只在本地适用的域名，这些域名是不会在全球的 DNS 服务器中出现的。更多信息请参考官方网站 www.thekelleys.org.uk。

DNSmasq 安装方便快捷，配置文件简单易懂，所以结合 DNSmasq 与 Puppet 来管理企业内网是最佳的选择。

1. DNSmasq 安装

首先到 DNSmasq 的官方网站下载最新版本的 DNSmasq。下载后通过源码编译方式安装 DNSmasq，安装完成后需要将 DNSmasq 安装目录配置文件 dnsmasq.conf.example 复制到 /etc/ 下。以下为详细的安装步骤，即下载、编译和安装，安装后可以将 DNSmasq 主配置文件复制到 /etc 目录下。

```
# wget http://www.thekelleys.org.uk/dnsmasq/dnsmasq-2.45.tar.gz
# tar -xvzf dnsmasq-2.45.tar.gz
# cd dnsmasq-2.45
# make && make install
# cp dnsmasq.conf.example  /etc/dnsmasq.conf
```

2. DNSmasq 的配置

编辑 /etc/dnsmasq.conf 的主配置文件的内容如下，根据如下配置信息，我们只要简单修改一下就可以使用 DNSmasq 了。

```
user=dnsmasq                                    # 启动账号
group=users                                     # 启动组号
interface=eth1                                  # 绑定网络接口
listen-address=192.168.0.1                      # 绑定 IP
bind-interfaces
resolv-file=/etc/resolv.conf                    # 域名解析文件
addn-hosts=/etc/hosts                           # 域名解析文件
```

3. 编辑域名解析文件

根据 /etc/dnsmasq.conf 配置文件中的 addn-hosts 参数，将域名解析配置文件改为系统默认定义的 /etc/hosts 文件，我们可以将 IP 和对应的虚拟域名追加到 /etc/hosts 文件中，来实现 DNSmasp 域名与 IP 映射关系，具体如下。

```
echo "192.168.0.1  www.example.com" >> /etc/hosts
```

4. 启动 DNSmasq

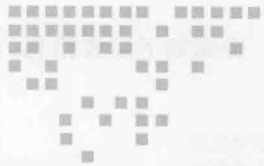
启动 DNSmasq 也是非常方便的，只需执行以下命令：

```
#  /usr/local/sbin/dnsmasq
```

启动后可通过 netstat 系统命令查看 DNSmasq 是否启动成功，如果 53 端口已成功启动，则说明 DNSmasq 已经正常工作。

```
localhost@#  netstat -tnl
tcp        0      0 192.168.0.1:53      0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53          0.0.0.0:*             LISTEN
tcp        0      0 fe80::a19:a6ff:fe27::53 :::*               LISTEN
tcp        0      0 ::1:53                  :::*
```

到目前为止已经成功启动了 DNSmasq，现在可以将 DNSmasq 服务器 IP 写入 Agent 的 resolv.conf 文件中，实现域名与 IP 的映射关系，DNSmasq 将作为一个内网域名解析系统而发挥作用。



Chapter 4 第 4 章

Puppet 目录结构、配置文件和命令详解

在第 3 章中介绍了在各系统环境下 Puppet 安装配置的方法，并介绍了与 Puppet 相关的版本控制和 DNS 等辅助工具的安装和配置，这些内容为后续章节知识的介绍奠定了基础。本章将从 Puppet 目录结构讲起，介绍目录结构和内部的配置文件，并通过案例来介绍这些配置文件的使用方法和作用，让读者对 Puppet 整体配置及目录结构有一个清晰的了解。然后按照命令来分类介绍 Puppet 的常用命令和参数，这会有效地降低读者的学习成本。我们首先讲解源码目录结构与主配置文件及目录结构；接着介绍 Puppet 的一些主配置文件的作用，配置文件的格式与案例；最后对 Puppet 命令进行了分类，按照分类讲解 Puppet 的命令、常用参数和应用案例。

4.1 源码与主配置文件的目录结构

本节主要介绍安装 Puppet 后源码目录结构和主配置文件及目录，让我们对整个 Puppet 的结构有一个大概的认识。这里以源码安装方式为例来介绍 Puppet 的目录结构，源码安装的优势是目录结构比较统一，便于介绍与讲解。运用源码安装方式需要将 Ruby、Facter 和 Puppet 安装到指定的目录中，如 /usr/local/puppet 目录。通常如果配置是 Agent 的话，Puppet 的源码安装后就基本可以使用了，但若是 Master 则还需要调整一下 /etc/puppet 目录的配置文件才能正常工作。要想调整目录中的配置文件，应该先对目录的结构有一个整体的理解。下面就来先了解一下源码目录结构，再了解主配置文件及目录的结构。

注意 RPM/yum 安装方式会将 Puppet 的命令安装到系统默认命令目录中，可以在 /sbin 和 /bin 目录中找到它们的源程序、配置文件与根目录，Puppet 开源社区版默认配置路径为 /etc/puppet 目录。

1. 源码目录结构

官方网站并不推荐通过源码编译安装 Puppet 及相关环境，但是由于一些企业内部网络系统安全限制，禁止通过 yum 从互联网安装软件包，所以了解源码安装还是有必要的。源码安装可以指定 Ruby、Facter 和 Puppet 的安装路径为统一目录，这样可以使后续 Puppet 源码迁移到其他机器更加方便。

Puppet 2.7.25 版本的源码方式安装目录结构和存放内容如下：

```
1  /usr/local/puppet
2   |_ bin/
3   |_    |_ erb  facter  filebucket  irb  pi  puppet  puppetdoc  ralsh  rdoc  ri
ruby  testrb
4   |_ sbin/
5   |_    |_puppetca  puppetd  puppetmasterd  puppetqd  puppetrun
6   |_ lib/
7   |_    |_ ruby
8   |_ share/
9   |_    |_man
```

下面简单分析一下上面的源码目录结构和相关的命令含义。

- 第 1 行：Ruby、Facter 和 Puppet 源码安装的根目录，目录位置由安装时的 configure 参数指定。
- 第 2 ～ 3 行：Ruby、Facter 和 Puppet 命令的存放目录。
- 第 4 ～ 5 行：系统管理员命令存放目录。
- 第 6 ～ 7 行：Ruby 的运行环境存放目录，另外还包含 Puppet 源码等。
- 第 8 ～ 9 行：Puppet 的帮助手册存放目录。

从第 2 ～ 5 行目录内存放命令介绍如表 4-1 所示。

表 4-1 Puppet 目录相关命令介绍

命 令	作 用	命 令	作 用
erb	erb 模板调试工具	ralsh	资源 RAL
facter	Agent 信息收集工具	rdoc	文档工具
filebucket	远程备份恢复工具	ri	文档工具
irb	Ruby 交互式 shell	ruby	Ruby 二进制程序
pi	资源帮助文档	testrb	Ruby 调试工具
puppet	Puppet 的命令集合	puppetca	证书签名授权命令
puppetdoc	文档生成工具	puppet	Agent 守护进程
puppetmasterd	Master 守护进程	puppetqd	Puppet 的队列
puppetrun	客户端更新工具		

2. 主配置文件及目录结构

Puppet 成功安装后默认开源社区版本主配置文件均存放在 /etc/puppet 目录。下面来看

一下 /etc/puppet/ 的目录结构。

主配置文件及目录结构如下：

```
1  /etc/puppet
2    |_  auth.conf
3    |_  autosign.conf
4    |_  files/
5    |_  fileserver.conf
6    |_  manifests/
7    |          |_ site.pp
8    |_ module/
9    |        |_apache
10   |        |      |_template/
11   |        |      |_mainfests/
12   |        |      |_init.pp
13   |        |      |_file/
14   |_puppet.conf
15   |_ ssl/
16   |_ tagmail.conf
17   |_ namespaceauth.conf
```

下面简单分析一下上面的目录结构和配置文件的作用。

- 第 1 行：/etc/puppet 目录是 Puppet 的主配置文件的根目录。
- 第 2 行：auth.conf 配置文件是 Agent 访问 Master 的权限认证文件，它的官方名称是 HTTP NetWork API，具体在 4.2.2 节介绍。
- 第 3 行：autosign.conf 配置文件是 Master 对 Agent 证书自动签名的配置文件，具体在 4.2.4 节介绍。
- 第 4 ～ 5 行：fileserver.conf 配置文件是 Master 向 Agent 同步静态文件的配置文件，具体在 4.2.5 节介绍。
- 第 6 ～ 7 行：Agent 入口的导航文件与逻辑文件等。此目录中存放的文件又称清单文件，manifests 目录与 module 目录是 Puppet 配置管理的两个重要的组成部分，具体在第 5 章介绍。
- 第 8 ～ 13 行：Puppet 的 module 目录又称基础模块，它的配置文件目录信息和结构具体在第 5 章介绍。
- 第 14 行：puppet.conf 配置文件是 Master 守护进程的主要配置文件，具体在 4.2.1 节介绍。
- 第 15 行：数字证书文件的缓存目录，Master 在此目录存放 CA 证书和已签名授权的 Agent 证书文件列表，Agent 在此目录存放被 Master 授权的证书文件。
- 第 16 行：tagmail.conf 文件是 Puppet 邮件发送配置文件，具体在 4.2.6 节介绍。
- 第 17 行：namespaceauth.conf 文件是名称空间配置文件，具体在 4.2.3 节介绍。

注意 无论通过什么方式安装 Puppet 的开源社区版本，它们的默认配置文件根路径都是 /etc/puppet 目录。

4.2 Puppet 主要配置文件介绍

通常我们使用 Puppet 的 C/S 架构工作模式。在 Puppet 安装后如果配置的机器是 Agent 就不需要太多的复杂配置，因为 Agent 可以守护进程方式运行或运行在 crontab 定时任务中。如果是初学 Puppet 我们通常将它运行在 crontab 定时任务中（一般在 crontab 定时任务中运行，Agent 无需配置文件）。但 Master 必须以守护进程方式运行在服务器上，这就需要先了解一些主要的配置文件的作用，才能掌握 Master。Master 的主要配置文件有 puppet.conf、auth.conf、namespaceauth.conf、autosign.conf、fileserver.conf 和 tagmail.conf。下面逐一介绍这些配置文件。

注意 介绍配置文件时主要以开源社区版本为例，如果情况特殊会进行注释。

4.2.1 puppet.conf 介绍

/etc/puppet/puppet.conf 配置文件（下称 puppet.conf）是 Master 的守护进程的主配置文件，文件中定义了 Master 的运行环境、启动加载文件、Puppet 的配置管理程序和授权 Agent 的证书目录等主要信息。守护进程启动前会根据此配置文件信息对系统环境进行预检，预检成功后守护进程才会启动。下面具体讲解各系统环境下 puppet.conf 配置文件的位置、文件格式和案例。

1. puppet.conf 配置文件的位置

为了保证本小节信息的完整性，笔者将介绍 UNIX/Linux 系统和微软 Windows 系统中不同版本的主配置 puppet.conf 文件的位置。建议初学 Puppet 的读者主要了解一下 UNIX/Linux 系统开源社区版本的 puppet.conf 文件位置即可。

1）UNIX/Linux 系统环境下 puppet.conf 配置文件存放的位置如下：

- Puppet 企业版本的配置文件存放位置是 /etc/puppetlabs/puppet 目录。
- Puppet 开源社区版本的配置文件存放的位置是 /etc/puppet 目录。

注意 在 UNIX/Linux 系统环境下，如果还是不确定 puppet.conf 配置文件的位置的话，可以通过 puppet agent --configprint confdir 命令来查看。

2）微软 Windows 系统环境下 puppet.conf 配置文件存放的位置如下：

- 微软 Windows 2003 系统配置文件的位置是 %ALLUSERSPROFILE%\PuppetLabs\puppet\etc 目录。
- 微软 Windows 7 系统配置文件的位置是 %PROGRAMDATA%\PuppetLabs\puppet\etc 目录。
- 微软 Windows 2008 系统配置文件的位置是 %PROGRAMDATA%\PuppetLabs\puppet\etc 目录。


注意 在微软 Windows 系列系统上 Puppet 配置文件支持使用 Windows 风格的“CRLF—Carriage-Return Line-Feed 回车换行符”作为结尾，UNIX/Linux 系列系统则要用 LF 风格结束符作为结尾。在微软 Windows 系列系统上 Puppet 企业版和开源社区版的配置文件位置是统一的目录。

2. puppet.conf 配置文件格式

puppet.conf 配置文件书写格式和 INI 文件格式非常相似，INI 格式文件由 3 个部分组成，分别是 [section] 节、setting = value 参数和“;”注解。其中 [section] 代表一组配置，setting=value 是组配置下的一个子集，可以写多个 [section] 节来标识多组配置。同样在 puppet.conf 文件中也有 [section] 节的概念，不过这里称呼它为区段。puppet.conf 配置文件被划为 3 个区段，每一区段用来配置 Puppet 的一个特定部分，[agent] 段用于 Agent 部分配置，[master] 段用于 Master 部分配置，最后还有一个 [main] 段用于 Puppet 的全局配置，Puppet 所有组件都遵循 [main] 段指定的配置。

来看一个 puppet.conf 配置文件的案例以加深理解，它包括的 3 个区段分别是 [main]、[master] 和 [agent]。分别来看一下这 3 个区段的内容作用，其中“#”后边的内容为注释。

1）[main] 区段用于 Puppet 的全局配置。

```
[main]
# 指定了 puppet 服务端的地址
server = puppet.example.com
# 是否实时刷新日志到磁盘
autoflush = false
# 日志目录
logdir = /var/log/puppet
# puppet 进程 pid 文件存放目录
rundir = /var/run/puppet
```

2）[Master] 区段用于 Puppet 的 Master 配置。

```
[master]
# 报告存放目录，客户端的每次执行会生成一份以日期 + 时间命名 yaml 文件报告
reportdir = /var/lib/puppet/reports
```

```
# 自动授权签名配置文件
autosign = true
autosign = /etc/puppet/autosign.conf
# puppetmaster 服务端监听地址
bindaddress = 0.0.0.0
# puppetmaster 服务端监听端口
masterport = 8140
# 通过此参数看到执行中的过程与变化
evaltrace = true
```

3）[agent] 区段用于 Puppet 的 Agent 配置。

```
# 客户端的名字
certname = puppet.example.com
# 是否后台运行,ture 表示后台运维
daemonize = true
# 是否允许证书自动覆盖，默认是不允许的，每个证书的有效期为 5 年
allow_duplicate_certs = true
# 是否上传客户端对 resouces 的执行结果（Puppet 2.7 以上版本此参数自动开启）
report = true
# 上传的方式
reports = store, http
# store 上传时的地址
report_server =  puppet.example.com
# store 上传时的端口
report_port = 8140
# http 上传时的地址
reporturl = http: //192.168.1.1: 3000/reports/upload
# 客户端执行间隔（20 分钟），如果不设置此参数默认为 30 分钟
runinterval = 20m
# 是否在执行时间上另加一个随机时间（0 到最大随机时间之间的一个整数值）
splay = true
# 加的随机时间的最大长度
splaylimit = 10m
# 客户端获取配置超时时间
configtimeout = 2m
# 日志记录是否加颜色
color = ansi
# 是否忽略本地缓存
ignorecache = true
```

注意 在 2.6.0 之前 puppet.conf 每一段名字都以程序命名，如 [master] 区段称为 [puppetmasterd]，[agent] 区段称为 [puppetd]。如果用户使用的是这种旧的配置格式文件，在启动更高版本的 Puppet 时系统会提示更新配置文件。

3. puppet.conf 配置文件案例

puppet.conf 是 Master 守护进程的主要配置文件，这里以启动 Master 守护进程的基

本参数为例来介绍 puppet.conf 配置文件。在修改 puppet.conf 配置文件前，首先通过 puppetmasterd --genconfig 命令来生成一份 puppet.conf 配置文件（Puppet3 以上版本已经不再提供 puppetmasterd 命令，所以可以使用 puppet master --genconfig 来生成 puppet.conf 文件）。默认生成的配置文件有 200 多个参数，但并不是所有参数都能用到，也不是所有参数都需要更改，很多参数可以保持其默认值。这里去掉一些没有必要的参数只保留启动 Master 守护进程的基本的参数。下面介绍如何通过这些基本参数来启动 Master 守护进程。关于 Master 的参数配置信息，更多信息请参考官方网站 http://docs.puppetlabs.com/references/latest/configuration.html。

以下是 puppet.conf 配置文件的常用配置参数。

```
1 [master]
2  bindaddress = 0.0.0.0
3  masterport = 8140
4  ssldir = /etc/puppet/server_ssl
5  logdir = /data1/puppet
6  manifest = /etc/puppet/manifests/site.pp
7  modulepath = /etc/puppet/modules
8  certname = puppet.example.com
9
10 [agent]
11 report = true
```

下面来分析一下 puppet.conf 配置文件的参数的含义。

- 第 1 行：[master] 段，主要配置 Master 的参数区域。
- 第 2 行：bindaddress 参数指定网卡接口，如果不设置此参数则默认绑定在 IP 地址 0.0.0.0 上。此参数通常用在特殊网络环境下配置监听指定的网卡。
- 第 3 行：masterport 参数是 Master 启动后绑定的端口，Puppet 启动端口默认绑定在 8140 上，如果有特殊需求，通过 masterport 参数可以修改 Puppet 的默认端口。
- 第 4 行：ssldir 参数是 Master 存放签名文件的配置路径。
- 第 5 行：logdir 参数是 Master 存放 Agent 上报日志的路径。
- 第 6 行：manifest 参数是 Agent 连接 Master 的起始配置的目录。
- 第 7 行：modulepath 参数是 modules 基础模块与配置文件的存放路径。
- 第 8 行：certname 参数可以设置 Master 的 FQDN（Fully Qualified Domain Name，中文翻译为“全称域名”），另外 certname 配置添加 FQDN 还可以解决证书的相关问题。
- 第 10 行：[agent] 段，其主要配置 Agent 的参数。
- 第 11 行：开启 Agent 的上报日志开关。

这里介绍了启动 Master 守护进程的最基本配置参数，可以参考 4.3.3 节介绍的 Master 启动方式来启动守护进程。

4.2.2 auth.conf 介绍

/etc/puppet/auth.conf 是配置文件（下称 auth.conf）。Master 本身是由 Web Server 提供 Agent 访问，如果没有权限控制，Agent 就可以遍历 Master 服务器上所有资源，这是非常不安全的。所以 auth.conf 配置文件主要负责 Agent 访问 Master 上一些目录和配置文件的权限和认证，它的官方名称是 HTTP Network API。为了网络访问安全，Agent 在访问 Master 过程中，其使用 HTTPS 协议进行通信交互，基本的访问格式如下：

```
https: //{server}: {port}/{environment}/{resource}/{key}
```

Master 会根据 auth.conf 配置文件来限制 Agent 的来源（来源包括 IP、域名或环境列表等）。限制访问 Master 某一个目录或一些目录的权限，有了 auth.conf 权限配置文件，就会使 Master 服务器更加安全。下面来看一下 auth.conf 配置文件的格式和配置文件案例，如图 4-1 所示。更多信息请参考官方网站 http：//docs.puppetlabs.com/guides/rest_api.html。

```
path [~] {/path/to/resource|regex}
[environment {list of environments}]
[method {list of methods}]
[auth[enthicated] {yes|no|on|off|any}]
[allow {hostname|certname|*}]
```

图 4-1 auth.conf 权限控制列表格式

注意 在旧的 Puppet 版本中，Puppet 使用了 XMLRPC 协议。XMLRPC 是使用 HTTP 协议作为传输协议的 RPC 机制，使用 XML 文本的方式，这种协议的工作效率比较低，在新版本中已经使用了 HTTP Newwork API 替代 XMLRPC 协议。旧版本中配置文件名是 authconfig，新版本中该功能已经被 auth.conf 配置文件所替代。

1. auth.conf 权限控制列表格式

图 4-1 所示是官方网站提供的 auth.conf 文件格式。

auth.conf 配置文件格式包含 7 个参数，分别是 path、environment、method、auth、allow、allow_ip 和 deny，每个参数有自己独立的值，通过这 7 个参数的自由组合，就形成了 Agent 访问 Master 目录权限控制的 ACL（Access Control Lists，权限控制列表）。下面分别来看一下 auth.conf 的 7 个参数。

1）path 参数：指定 ACL 的路径。path 后接系统路径、正则表达式、路径前缀和资源等信息。

2）enviroment 参数：它可以包含一个环境或多个环境列表，如果不设置则默认为所有环境。Puppet 的环境主要用于“灰度”功能，在第 5 章详细介绍 Puppet 的环境。

3）method 参数：包含 find（查找）、search（搜索）、save（保存）和 destroy（销毁），可以在 method 后设置任意一个参数或用逗号做分隔设置多参数，默认是设置所有参数。

4）auth 参数：auth 参数包含 yes 或 on、any 和 no 或 off，不同的参数可以匹配不同的请求，auth 参数默认值为 yes 或 on，下面分别看一下这几个参数的含义。

- auth 设置为 yes 或 on[⊖]均表示匹配那些已经通过认证的 Agent 请求。
- auth 设置为 any 意味着只匹配认证进行中和没有被认证的请求。
- auth 设置 no 或 off，均表示匹配未认证过的 Agent 请求。认证过的请求将会跳过此 ACL。

5）allow 参数：它的值可以是 hostname 或 certname。Puppet 2.7.1 以后的版本还支持 Perl 或 Ruby 的正则表达式，如 allow /^[\w-]+.example.com$/ 通过正则表达式来匹配 hostname 或 certname。

6）allow_ip 参数：它后接一个 IP 或 IP 的网段，表示允许指定的 IP 范围通过。

7）deny 参数：它后接一个 IP、多个 IP、网段或域名等，表示禁止这些范围访问 Master 的目录权限。

2. auth.conf 配置文件案例

auth.conf 是 Agent 访问 Master 的目录时 ACL 权限控制列表配置文件，Puppet 默认提供的 auth.conf 模板文件已经为 /etc/puppet 目录配置好了 ACL，只有特殊的需求时才会更改它。下面让通过 4 个案例修改 auth.conf 配置文件，看 Master 是如何对 Agent 做 ACL 权限控制的。可以编辑 auth.conf 配置文件追加以下内容。

案例 1

限定 Agent 访问 facts 路径，只准许来自 puppet1.example.com 和 puppet2.example.com 的域访问。具体设置方法如下：

```
path /facts
method find, search
auth yes
allow puppet1.example.com, puppet2.example.com
```

- path 参数：后面接被访问路径。
- method 参数：find，search 表示准许访问 facts 路径下的文件。
- auth 参数：匹配已经通过认证请求的 Agent。
- allow 参数：准许被访问的域。

案例 2

限定已经通过了认证请求的 Agent 并且 IP 范围是 192.168.1.0/24，在这个范围内可以访

⊖ yes 与 on 参数结果是一致的，均表示开启或认证通过的含义。在 Puppet 中我们会看到很多类似的情况，同一个参数有多个值含义相同，如 auth.conf 文件中 auth 参数的 no 与 off 值含义相同结果一致。官方文档中并没有过多解释参数值相同的原因，但笔者猜测更多的原因是因为设计初没有考虑太周到，后续的更改为了与之前版本的兼容而导致的结果。

问 /file_metadata/user_files/ 和 /file_content/user_files/ 两个路径。具体的设置方法如下：

```
path ~ ^/file_(metadata|content)/user_files/
auth yes
allow_ip 192.168.100.0/24
```

- path 参数：通过正则表达式匹配 /file_metadata/user_files/ 和 /file_content/user_files/ 两个路径。
- auth 参数：匹配已经通过认证请求的 Agent。
- allow_ip 参数：准许 IP 范围是 192.168.100.0/24 的 Agent 访问。

案例 3

禁止 guest.puppet.com 域访问“/”根路径。具体设置方法如下：

```
path /
deny guest.puppet.com
```

- path 参数：匹配根路径。
- deny 参数：禁止 guest.puppet.com 域访问。

案例 4

auth.conf 的危险模式，此模式通常用于测试。具体设置方法如下：

```
path /
auth any
allow *
```

- path 参数：匹配根路径。
- auth 参数：匹配认证进行中和没有被认证的请求。
- allow 参数：“*”表示准许所有的访问来源。

提示 Puppet 源码安装或 RPM 安装方式都会提供 auth.conf 模板文件，源码安装 auth.conf 模板文件放在源码安装目录里，RPM 安装 auth.conf 模板文件安装到 /etc/puppet 目录里。

4.2.3 namespaceauth.conf 介绍

/etc/puppet/namespaceauth.conf 配置文件（下称 namespaceauth.conf）用于指定访问 Puppet 的名称空间。namespaceauth.conf 文件并非 Puppet 中的必要文件，当 puppet.conf 文件中 listen=true 时才需要用到此配置文件，否则 Puppet 会报 will not start without authorization file namespaceauth.conf 这样的错误。以下为 namespaceauth.conf 配置内容，namespaceauth.conf 配置内容的具体作用在 4.3.15 节通过 puppet kick 工具介绍。

```
[fileserver]
```

```
  allow *.domain.com
[puppetmaster]
  allow *.domain.com
[puppetrunner]
  allow culain.domain.com
[puppetbucket]
  allow *.domain.com
[puppetreports]
  allow *.domain.com
[resource]
  allow server.domain.com
```

4.2.4 autosign.conf 介绍

/etc/puppet/autosign.conf 配置文件（下称 autosign.conf）主要用于自动签名证书功能，需要管理员手工创建。默认 Agent 访问 Master 时管理员需要手动授权 Agent 证书签名，通过 Master 上的 autosign.conf 配置文件，可以实现对某一来源或所有来源的 Agent 做自动授权证书签名。下面来看一下 autosign.conf 配置文件案例。

autosign.conf 文件授权签名方式有如下两种。

方式一：在 Master 中追加“*”通配符到 autosign.conf 配置文件，表示不做任何限制，授权所有的 Agent 访问 Master 时会自动授权证书签名。

```
# cat /etc/puppet/autosign.conf
*
```

注意 autosign.conf 配置文件不做任何限制，仅供我们学习使用。如果在工作环境中使用，建议在这里要做严格的限制，否则会引发一些安全问题，请读者慎重使用。

方式二：在 Master 中追加 *.example.com 到 autosign.conf 配置文件，表示只有 *.example.com 域的 Agent 访问 Master 时会自动授权证书签名。

```
# cat /etc/puppet/autosign.conf
*.example.com
```

添加方式以后再次从 Agent 上访问 Master，这次 Master 的响应信息会自动签名来自各域名的证书文件，如图 4-2 所示。

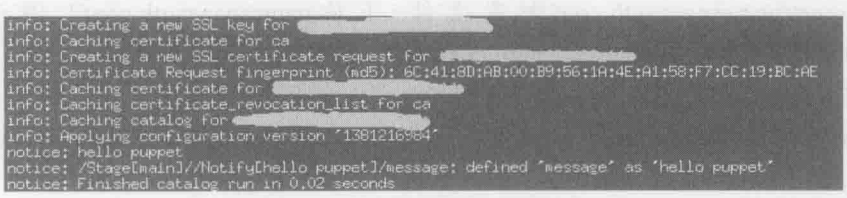


图 4-2 autosign.conf 自动授权

4.2.5 fileserver.conf介绍

/etc/puppet/fileserver.conf配置文件（下称fileserver.conf）是Master目录的挂载配置文件，它包含了Master的挂载目录位置和挂载目录的授权信息等。与其他配置文件相比，fileserver.conf并非是Puppet必要的配置文件，只有在Agent服务器从Master获取一个文件或文件列表时才会用到它。以下为fileserver.conf配置模板文件。

```
# fileserver.conf
[mount_point]
path /path/to/files
allow *.example.com
deny *.wireless.example.com
```


Puppet源码安装或RPM安装方式都会提供fileserver.conf模板文件，以源码的方式安装fileserver.conf，模板文件会被安装到源码安装目录里，通过RPM安装fileserver.conf，模板文件会被安装到/etc/puppet目录里。

在filesserver.conf配置文件中，[mount_point]为挂载点；path参数定义操作系统挂载目录的根路径为/path/to/files目录；allow参数准许来自*.example.com域访问Master的挂载目录；最后通过deny参数禁止来自*.wireless.example.com域访问Master的挂载目录，这就是Puppet Fileserver的基本配置。来看一个例子，从Master上同步sudoers文件到Agent的/etc/sudoers，通过file资源同步静态文件。

```
file{"/etc/sudoers":
 mode => 440,
 owner => root,
 group => root,
 source=>"puppet:///mount_point/sudoers",  # 指定Master服务器上sudoers文件存放位置
}
```

Agent通过Master上的file资源将sudoers静态文件按指定的系统权限和系统用户信息同步到Agent的/etc/目录下，这里在同步静态配置文件时需要在file资源中写明source属性，并指定要同步给Agent的sudoers配置文件在Master上的位置，Agent在访问Master时会根据file资源中的配置信息进行同步。在source属性中的puppet:///的根路径其实就是/etc/puppet/fileserver.conf中Path参数的挂载目录路径。

通过Puppet的fileserver.conf向Agent同步数据时，建议不要同步大于1MB的数据文件，Puppet虽然有实现同步静态文件的功能，但其并未使用专业的文件同步数据协议，如果多个Agent同时同步比较大的数据，就会引起Master超时，最终导致同步失败。目前同步数据可以使用的协议只有Puppet文件服务协议一种，在Puppet未来的版本中，文件服务器会支持更多协议，如HTTP或rsync协议等。

4.2.6 tagmail.conf 介绍

/etc/puppet/tagmail.conf 配置文件（下称 tagmail.conf）是 Puppet 发送邮件程序的配置文件。Master 支持在检测到某一资源发生变化时，通过邮件通知一个或多个系统管理员的功能。通过修改 Master 和 Agent 上的配置文件就可以实现这些需求。开启邮件配置功能分为以下 5 个步骤。

步骤 1　在 Agent 的 puppet.conf 中设置 report = true，打开上报开关（此参数在 2.7 版本以上为自动打开）。

步骤 2　在 Master 的 puppet.conf 中设置 reports = tagmail，设置 tagmial 报告处理器。关于 Puppet 的报告，会在第 12 章详细介绍。

步骤 3　在 Master 的 puppet.conf 中设置 reportfrom 值，用来发送邮件的邮箱名称，默认是“Report@ 系统 FQDN 名”。同时还要设置发送邮件的服务器 smtpserver 或 sendmail。

- sendmail 参数：默认位置为 /usr/sbin/sendmail，通过 sendmail 参数可以覆盖此位置值。
- smtpserver 参数：设置 stmp 服务器的地址，默认为空。

步骤 4　在 Master 的 puppet.conf 中设置 tagmap = /etc/puppet/tagmail.conf。其实刚刚我们也提到过 tagmail.conf 默认配置文件位置就在 /etc/puppet/ 下，如果有特殊需求需要更改 tagmail.conf 路径，可以通过修改 puppet.conf 参数来实现。

步骤 5　将管理员邮箱追加到 tagmail.conf 中，如图 4-3 所示。

```
all: log-archive@example.com
webserver, !mailserver: httpadmins@example.com
emerg, crit: james@example.com, zach@example.com, ben@example.com
```

图 4-3　tagmail.conf 配置文件

Puppet 源码安装或 RPM 安装方式都会提供 tagmail.conf 模板文件，源码安装 tagmail.conf 模板文件会放在源码安装目录里，RPM 安装 tagmail.conf 模板文件会被安装到 /etc/puppet 目录里。

4.3 Puppet 命令详解

由于不同版本之间的命令和参数稍微有一些区别，为了规避这些版本间的区别带来的问题，笔者以 Puppet 2.7.* 版本分支为例来介绍 Puppet 命令的使用状况。首先看一下 Puppet 命令的历史，了解一下它的前世今生，然后按照 Puppet 命令的使用频率了解它的分类，最后根据命令分类来分别了解 Puppet 命令。

4.3.1 Puppet 命令的前世今生

在 Puppet 中完成一个任务可以由多种命令实现，这也是历史原因导致的。在开始的 Puppet 版本中，它将所有的功能放置到不同的二进制中来实现；2.6.0 之后的版本，Puppet 将所有功能集成到了一个单独的二进制程序，即 puppet。我们先来对独立命令和集成命令进行一些比较，具体如表 4-2 所示。

表 4-2 独立命令与集成命令对照表

独立命令	作 用	集成命令	作 用
pi	资源帮助文档	puppet describe	资源帮助文档
puppetdoc	文档生成工具	puppet doc	文档生成工具
puppetmasterd	守护进程	puppet master	Master 命令
puppetrun	客户端更新工具	puppet kick	客户端主动更新工具
puppetqd	队列命令	puppet queue	队列命令
puppet	Agent 守护进程	puppet agent	Agent 命令
puppetca	证书文件授权命令	puppet cert	证书文件授权命令
filebucket	远程备份恢复工具	puppet filebucket	远程备份恢复工具
ralsh	Puppet 资源抽象工具	puppet resource	Puppet 资源抽象工具

以 Puppet 的 Master 系统命令为例，可以使用 puppet master 的方式启动 Master 守护进程（其使用方式类似于 Git 版本控制工具），也可以使用单独的 puppetmasterd 的方式启动守护进程，其参数和执行最终的结果都是一样的。这里推荐使用 puppet 集成命令方式，因为这是今后 Puppet 的发展趋势。


注意 在 Puppet 3 版本中，独立命令功能将不再提供。

4.3.2 如何掌握 Puppet 命令

Puppet 为我们提供了丰富的命令来管理配置服务器。截至 Puppet 2.7.* 版本分支，Puppet 共集成了 34 个命令与子命令。首先来介绍一下 Puppet 共集成了哪些命令与子命令，然后介绍如何查看子命令的参数，最后按照命令的功能和频率划分，将它们分为 4 大类来分别介绍。

1. Puppet 命令与子命令

输入 puppet help 命令查看 puppet 命令与子命令的帮助文档，结果如下：

```
# puppet help
Usage: puppet <subcommand> [options] <action> [options]
Available subcommands, from Puppet Faces:
  # 子命令
```

```
  ca                                  管理本地证书文件
  catalog                             编译、保存、查看和转换 catalog
  certificate                         提供接入 CA 证书管理
  certificate_request                 管理证书请求
  certificate_revocation_list  Manage 列表显示被删除的证书
  config                              配置选项
  facts                               存储 facts 返回信息
  file                                在 filebucket 中文件个数和存储文件
  help                                显示帮助信息
  instrumentation_data                管理监听数据
  instrumentation_listener            管理监听状态
  instrumentation_probe               管理监听探测
  key                                 创建、保存和删除证书的密钥文件
  man                                 显示帮助手册信息
  module                              通过 Puppet Forge 创建、安装和查找基础模块
  node                                查看与管理节点
  parser                              *.pp 文件语法检查
  plugin                              插件管理
  report                              创建、显示和提交报告
  resource                            资源 RAL，仅供 API 使用
  resource_type                       查看类、默认资源与来自 manifests 的节点信息
  secret_agent                        模拟 Agent
  status                              查看服务状态
Available applications, soon to be ported to Faces:
  # 命令
  agent                               Puppet 守护进程
  apply                               Puppet 代码应用工具
  cert                                管理 Puppet 认证
  describe                            查看资源的使用工具
  device                              远程网络设备管理
  doc                                 文档生成工具
  filebucket                          文件恢复与还原
  inspect                             发送报告
  kick                                Agent 主动更新工具
  master                              Master 守护进程
  queue                               Puppet 队列
See 'puppet help <subcommand> <action>' for help on a specific subcommand action.
See 'puppet help <subcommand>' for help on a specific subcommand.
```

2. 查看子命令属性

Puppet help 命令输出包含了子命令与命令，Puppet 子命令通常可以协助命令来完成任务。查看子命令的使用方法如下：

```
# puppet help ca
USAGE: puppet ca <action>
This provides local management of the Puppet Certificate Authority.
OPTIONS:
  --mode MODE                         - The run mode to use (user, agent, or master).
```

```
    --render-as FORMAT          - The rendering format to use.
    --verbose                   - Whether to log verbosely.
    --debug                     - Whether to log debug information.
ACTIONS:
    Destroy                 undocumented action
    Fingerprint             undocumented action
    Generate                undocumented action
    List                    List certificates and/or certificate requests.
    Print                   undocumented action
    Revoke                  undocumented action
    Sign                    undocumented action
    Verify                  undocumented action
```

后续章节会通过案例来分别介绍 Puppet 子命令的使用方式，这里就不详细介绍了。

3. Puppet 命令种类划分

Puppet 命令与子命令按功能和频率划分可以分为 4 类，如表 4-3 所示。下面对其中比较重要的命令进行介绍，其他的命令在后续章节用到时会简单介绍。后续章节用不到的命令，其实使用频率也就非常低了，限于篇幅，本书不再介绍。

关于 Puppet 命令与子命令的更多信息请参考官方网站 http://docs.puppetlabs.com/man/ 或 http://docs.puppetlabs.com/references/2.7.latest/man/index.html。

表 4-3 Puppet 命令

分　类	命　令
核心命令	puppet master、puppet agent、puppet apply、puppet cert、puppet module、puppet resource
常用命令	puppet describe、puppet device、puppet doc、puppet help、puppet man、puppet node、puppet parser、puppet plugin
不常用命令	puppet ca、puppet catalog、puppet certificate、puppet certificate_request、puppet config、puppet facts、puppet status puppet certificate_revocation_list、puppet file、puppet filebucket、puppet inspect、puppet secret_agent、puppet report puppet instrumentation_data、puppet instrumentation_listener、puppet resource_type、puppet instrumentation_probe、puppet key
即将废弃命令	puppet kick、puppet queue

4.3.3 puppet master 介绍

puppet master 是 Master 守护进程命令，启动后默认以 TCP 协议监听在 8140 端口上。它的工作原理是通过 Ruby 的 Webrick Web 接收 Agent 服务器的请求，根据请求内容与 Master 上的 site.pp 文件进行匹配，并根据 site.pp 文件内容编译成 catalog 向 Agent 分发，在 Agent 上应用这些配置信息。其实 Master 的守护进程还有另外一种启动形式，就是通过 service puppetmasterd start 启动。这种方式启动简单，但是出现问题不容易发现，如 puppet.conf 文件不存在，守护进程是无法启动的，而 service puppetmasterd start 命令既不会报错

也不会启动守护进程，初学者可能就会比较迷惑。所以这里推荐以 puppet master 作为启动命令，因为它会反馈更多的启动信息，让我们知道它的启动过程，并在启动失败时，也可了解失败的原因。下面来看一下 puppet master 的常用参数和案例。

1. puppet master 常用参数

以下为 puppet master 的常用参数，此命令由 Puppet 的作者 Luke Kanies 提供。

```
1  --daemonize
2  --no-daemonize
3  --debug
4  --logdest
5  --verbose
6  --version
7  --compile
8 --masterport
9 --pidfile
10 --servertype
```

下面简单分析一下 puppet master 命令的常用参数。

- daemonize 参数：将进程发送到后台运行，是 Master 的默认参数。
- no-daemonize 参数：将进程输出信息发送到标准输出。
- debug 参数：打开调试信息。
- logdest 参数：指定输出日志的路径和文件名。它可以输出到屏幕终端、系统日志和指定文件中，默认是发送到屏幕终端。
- verbose 参数：输出扩展信息。
- version 参数：显示 Master 的版本信息。
- compile 参数：将编译后的 catalog 以 JSON 格式输出到 $vardir/yaml/ 目录下。我们也可以通过 puppet master --complie example.puppet.com 方式，其中 example.puppet.com 表示 Agent 的 Hostname，它的输出结果为 JSON 格式数据，将它导入 puppet apply 中使用。
- masterport 参数：Master 自定义端口。
- pidfile 参数：Master 在监听多端口时的 pid 文件不能公用，所以要通过 pidfile 参数指定不同的 pid 文件位置。
- servertype 参数：Master 的启动方式。默认设置为 WEBRick，也可以设置为 Mongrel 方式，Mongrel 方式启动方式会在第 11 章详细介绍（需要注意的是，Mongrel 方式启动适用于 Puppet 2.7 以下的版本，Puppet 3 已经取消了 Mongrel 方式启动）。

2. puppet master 案例

通过 puppet master 命令和参数方式启动 Master，启动前先要确认 puppet.conf 配置文件

是否存在，并且按照机器状况进行配置。通过 puppet master 与参数方式启动 Puppet 守护进程的方法如下。

```
# puppet master --verbose --no-daemonize >> master.log 1>&2 &
```

通过 verbose 参数使 Master 输出详细的日志，通过 no-daemonize 参数使 Master 在终端运行，并将进程信息输出重定向到标准输出。通过这种方式启动可以看到 Puppet 的一些初始化过程如图 4-4 所示。当启动失败时会输出错误原因以方便我们解决问题。如果不是首次启动的话，则推荐在命令后面增加“ >> master.log 1>&2 &”符号，让它在后台运行，而日志输出到 master.log 文件中。

```
notice: Starting Puppet master version 2.7.21
info: mount[files]: allowing * access

info: access[^/catalog/([^/]+)$]: allowing 'method' find
info: access[^/catalog/([^/]+)$]: allowing $1 access
info: access[^/node/([^/]+)$]: allowing 'method' find
info: access[^/node/([^/]+)$]: allowing $1 access
info: access[/certificate_revocation_list/ca]: allowing 'method' find
info: access[/certificate_revocation_list/ca]: allowing * access
info: access[^/report/([^/]+)$]: allowing 'method' save
info: access[^/report/([^/]+)$]: allowing $1 access
info: access[/file]: allowing * access
info: access[/certificate/ca]: adding authentication any
info: access[/certificate/ca]: allowing 'method' find
info: access[/certificate/ca]: allowing * access
info: access[/certificate/]: adding authentication any
info: access[/certificate/]: allowing 'method' find
info: access[/certificate/]: allowing * access
info: access[/certificate_request]: adding authentication any
info: access[/certificate_request]: allowing 'method' find
info: access[/certificate_request]: allowing 'method' save
info: access[/certificate_request]: allowing * access
info: access[/]: adding authentication any
info: access[/tmp]: adding authentication any
info: access[/tmp]: allowing 'method' find
info: access[/tmp]: allowing 'method' save
info: access[/tmp]: allowing * access
info: Inserting default '/status' (auth true) ACL because none were found in '/etc/puppet/auth.conf'
```

图 4-4 Master 守护进程输出信息

4.3.4 puppet agent 介绍

puppet agent 是 Agent 访问 Master 获取配置信息的命令，下面来看一下它的常用参数和案例。

1. puppet agent 常用参数

以下为 puppet agent 的常用参数，puppet agent 命令由 Puppet 的作者 Luke Kanies 提供。

```
1  --daemonize
2  --no-daemonize
3  --debug
4  --disable
```

```
5  --enable
6  --noop
7  --onetime
8  --server
9  --test
10 --verbose
11 --version
12 --waitforcert
13 --environments
```

下面我们简单分析一下 puppet agent 的常用参数。

- daemonzie 参数：将进程输出信息发送到后台，其为 Agent 的默认参数。
- no-daemonize 参数：将进程输出信息发送到标准输出。
- debug 参数：打开 Puppet 调试模式，输出启动过程中的调试信息。
- disable 参数：它会在 Agent 机器上建立一个锁定文件，在锁定文件没有删除之前，Agent 并不会将 Master 获取的配置信息应用到本机，此命令可以防止 Agent 守护进程重复执行。
- enable 参数：通过此参数可以清理锁定文件，并恢复 Puppet 的正常工作状态。
- noop 参数：此参数又被称为 Puppet 的 dry run 模式，在此模式下运行主机不会做出任何变更，但是它会告诉你本次执行中 Puppet 都做了些什么，根据 dry run 模式输出结果来查看是否和我们的预期一致。
- ontime 参数：可以让 Puppet Agent 运行一次就停止。
- server 参数：可以指定 Master 的域名。
- test 参数：一些命令的集合，通过 --test 可以打开 onetime、verbose、ignorecache、no-daemonize、no-usecacheonfailure、detailed-exit-codes、no-splay 和 show diff 等参数。
- verbose 参数：输出 Puppet 扩展信息。
- version 参数：显示 Puppet 版本信息。
- waitforcert 参数：Agent 访问 Master 时的证书签名的等待时间（单位为秒），在没有得到 Master 授权证书签名的情况下，Puppet Agent 守护进程默认每两分钟向 Master 请求发送一次证书并等待响应。如果 waitforcert 参数值设置为 0 秒，则表示不等待。
- environments 参数：后面接 Puppet 环境参数，此功能主要用于配置文件的灰度。第 5 章会详细介绍。

2. puppet agent 案例

以下是 puppet agent 的两个案例，案例 1 主要介绍 puppet agent 如何访问 Master，案例 2 介绍 Puppet 的 dry run 模式。

案例 1

让我们来看一下 Agent 是如何通过最基本的参数来访问 Master 的。首先通过 puppet

agent 后接 test 参数的方式来访问 Master 获取配置信息。具体命令如下：

```
# puppet agent --server=puppet.example.com --test
```

Agent 访问 Master 成功后会显示如下信息：

```
info: Caching catalog for agent
info: Applying configuration version '1382198703'
notice: hello puppet
notice: /Stage[main]//Notify[hello puppet]/message: defined 'message' as 'hello puppet'
notice: Finished catalog run in 0.02 seconds
```

输出的结果包含了本次访问 Master 的时间，获取的应用信息和整个运行的耗时。这些信息有助于我们了解本次 Agent 最终结果是否成功。

注意 Agent 可以以两种方式运行，一种方式是命令接参数直接连接 Master；另一种方式是以守护进程的形式在 UNIX/Linux 后台运行，默认每 30 分钟访问一次 Master，但是这样并不灵活。在此推荐读者将 Agent 的命令放到 crontab 中运行。

案例 2

我们通过 Puppet 来管理服务器时最怕的是什么？最怕的就是误操作，误操作会带来毁灭性的打击，一个误操作可能导致一天的努力付诸东流，所以通过 Puppet 管理配置服务器要谨慎再谨慎。那么如何避免错误的发生呢？第 5 章会介绍 puppet 的环境，可以将它理解为测试中的灰度，通过它可以降低误操作对线上的影响。可能读者不禁要问，除了环境就没有别的办法了吗？办法当然也是有的，通过 puppet agent 命令的 noop 参数就可以实现。通常 noop 参数又称为 dry run 模式，即模拟执行。在此模式下，puppet 的 Agent 和往常一样也会执行一次，但是它并不会改变 Agent 的系统状态。让我们看一下它的使用方法和最终结果。

定义 httpd::install 类，它的功能是同步 Master 上的 httpd.conf 配置文件到 Agent，我们修改 http::install 类中已经声明的 $serveradmin 变量，将内容由 admin@qq.com 改为 test@qq.com。具体代码如下：

```
class httpd::install {
$listenaddress = "$::ipaddress"
$serveradmin = "admin@qq.com"      # 更改为 test@qq.com
$user = "nobody"
$group = "nobody"
   file {'/usr/local/apache2/conf/httpd.conf':
    mode => '777',
   owner => 'root',
   group => 'root',
 content => template("httpd/httpd.conf.default.reb")
   }
```

在 Agent 访问 Master 时追加 noop 参数。具体命令如下：

```
# puppet agent --server=puppet.example.com --test --noop
notice: /Stage[main]/Httpd: : Install/File[/usr/local/apache2/conf/httpd.conf]/content:
--- /usr/local/apache2/conf/httpd.conf  2013-07-05 10: 48: 52.000000000 +0800
+++ /tmp/puppet-file20131022-13983-1yo2o84-0    2013-12-12 14: 50: 17.000000000 +0800
@@ -277,7 +277,7 @@
 # e-mailed.  This address appears on some server-generated pages, such
 # as error documents.  e.g. admin@your-domain.com
 #
-ServerAdmin admin@qq.com      # 本次变更前内容
+ServerAdmin test@qq.com       # 本次变更后内容
```

执行后我们可以看到 Agent 上的 httpd.conf 配置文件内容并没有变化，响应结果中会显示本次的变更内容，但是它并不会真的修改 Agent 上的 httpd.conf 配置文件，这就是 dry run 模式的好处，我们可以知道它的变更结果，但又并不会真的更改文件。重要配置文件变革前，可以尝试 noop 参数，并确认是否与我们的预期一致，从而达到万无一失。

4.3.5 puppet cert 介绍

puppet cert 是管理 Puppet 的证书签名的命令，在 Agent 访问 Master 时使用的是 SSL 安全套接字，其优点是加密双方的通信数据，从而保证信息安全。首次访问需要 Master 对 Agent 的证书签名才可以正常建立 SSL 通信，所以首次连接需要先通过 puppet cert 命令在 Master 上对 Agent 的数字证书签名。我们通过 puppet cert 命令还可以实现管理、授权、回收、显示和产生签名文件。以 puppet cert 方式为例，下面来看一下它的常用参数和案例。

1. puppet cert 常用参数

以下为 puppet cert 的常用参数，此命令由 Puppet 的作者 Luke Kanies 提供。

```
1  --clean
2  --generate
3  --list
4  --print
5  --revoke
6  --sign
7  --verify
```

下面来简单分析一下 puppet cert 命令的常用参数。

- clean 参数：清理 Master 主机上存储的所有相关证书文件。
- generate 参数：为指定域名授权签发一个证书文件。
- list 参数：在 Master 上可以列出目前 Agent 机器等待签发证书的信息。
- print 参数：打印证书的版本信息。
- revoke 参数：回收指定 Agent 的证书。

- verify 参数：确认证书是否由本地 CA 签发。

2. puppet cert 案例

这里介绍 puppet cert 的两个案例，案例 1 主要介绍 puppet cert 如何授权 Agent 证书签名；案例 2 介绍预授权签名证书文件。

案例 1

通过前面的介绍我们已经了解了 puppet cert 的用途，下面来看一下在 Puppet 的 C/S 架构下，Master 是如何为 Agent 签发证书文件的。首先在 Master 上通过 puppet cert list 命令显示目前都有哪些 Agent 等待签发证书。命令如下：

```
# puppet cert list
example.web.puppet.com
```

可以看到有一台 HOSTNAME 为 example.web.puppet.com 的 Agent 在等待签发证书文件，这时可以通过 sign 参数对等待的证书进行签名，授权签名后的 Agent 才可以建立 SSL 通信并访问 Master。我们来对 HOSTNAME 为 example.web.puppet.com 的 Agent 授权签名证书文件，命令如下：

```
# puppet cert sign example.web.puppet.com
```

通过 sign 参数签名后的 HOSTNAME 为 example.web.puppet.com 的 Agent 就可以正常访问 Master 了。这里只看到了一台要签名的 Agent，但如果要签名的 Agent 比较多怎么办？需要一个个签名吗？大家不用担心，puppet cert 还提供了 all 参数，可以通过它对所有等待的 Agent 进行手动签名。在等待签名的 Agent 服务器比较多的情况下，可以使用 all 参数，具体如下：

```
# puppet cert sign --all
```

Agent 被签名后，其生成的签名证书文件会根据 Master 的 puppet.conf 文件中的 ssldir 参数的指定路径进行存储，参数默认会将生成的证书文件存放在 /var/lib/puppet/ssl/signed/example.web.puppet.com.pem。

案例 2

还有一种相对安全的签名方式，即“预签名方式”，也就是管理员提前将签名证书文件生成后推送到 Agent 机器上，通过 puppet cert --generate 后接 HOSTNAME 的方式来预生成签名证书。命令如下：

```
# puppet cert --generate example1.puppet.com
```

这里它会预生成 example1.puppet.com 证书，包括 Agent 的私钥、Agent 的证书和 CA 证书。具体如下：

```
/etc/puppet/ssl/private_keys/example1.puppet.com.pem
```

```
/etc/puppet/ssl/certs/example1.puppet.com.pem
/etc/puppet/ssl/certs/ca.pem
```

传输这 3 个所需的证书文件到新的 Agent 上，这样就可以省略证书请求这一步了。对于那些批量接入 Puppet 的 Agent，这样的方式相对要更安全一些。

提示 除了对每一个证书文件手动进行证书签名外，还可以参考 4.2.4 节的内容进行自动签名，在这种模式下，从指定 IP 范围或域名发起的访问都会被自动签名。不过这一操作确实存在一些安全问题，建议读者最好在熟悉了 Puppet 之后再使用这一特性。

4.3.6 puppet apply 介绍

puppet apply 是一个单独执行代码的工具，可以在本机运行以 .pp 结尾的程序（.pp 扩展名是 Puppet 配置管理语言的源文件）。之前我们介绍过 Puppet 的两种运行方式，一种是 C/S 架构运行，另一种就是单机运行，而 puppet apply 就是单机运行的工具，通常 puppet apply 适用于代码调试环节。下面来看一下 puppet apply 的常用参数和案例。

1. puppet apply 常用参数

以下为 puppet apply 的常用参数，puppet apply 由 Puppet 的作者 Luke Kanies 提供。

```
1 --debug
2 --logdest
3 --execute
4 --verbose
5 --apply
6 --catalog
```

下面简单分析一下 puppet apply 工具的常用参数。

- debug 参数：打开 Puppet 调试模式，输出执行过程中的调试信息。
- logdest 参数：puppet apply 工具在执行 *.pp 代码过程中，默认会将信息输出到屏幕终端，通过此参数可以指定输出的路径或文件来改变日志输出方式。
- execute 参数：用于执行 Puppet 代码片段。
- verbose 参数：输出扩展信息。
- apply 参数和 catalog 参数：都可以接 JSON 格式的文件或从标准输入导入 JSON 格式的数据，不过目前 catalog 参数已经取代 apply 参数的功能。注意，通过 puppet master --compile 可以生成 JSON 格式的数据，然后通过 puppet apply 工具实现 catalog 参数接 JSON 格式数据。

2. puppet apply 案例

以下是 puppet apply 的两个案例。案例 1 主要介绍 puppet apply 如何使用，案例 2 介绍

通过 puppet apply 参数形式执行 Puppet 代码。

案例 1

通过此案例可以了解 puppet apply 工具是如何执行 .pp 文件的。首先创建 test.pp 文件，追加 notify 资源与“hello world”值到文件中。命令如下：

```
# cat test.pp
notify{"hello world": }
```

notify 资源的作用是将信息输出到屏幕终端，可以看到，它与 shell 语言的 echo 非常相似，即把结果打印出来。通过 puppet apply test.pp 可以看到执行结果。

```
# puppet apply test.pp
notice: hello world    # notify 资源的输出
notice: /Stage[main]//Notify[hello world]/message: defined 'message' as 'hello world'
notice: Finished catalog run in 0.01 seconds
```

通过 notify 资源将 test.pp 中的“hello world”信息发送到系统标准输出，同时显示调用 notify 资源的执行时间，这和我们 C/S 架构执行的结果是一致的。

案例 2

我们还可以通过 puppet apply 工具的 execute 参数直接调用 Puppet 代码片段，为了节约篇幅这里仍然通过 notify 资源来介绍 execute 参数的使用。具体如下：

```
# puppet apply --execute "notify{"hello world"}"
notice: hello world    # notify 资源的输出
notice: /Stage[main]//Notify[hello world]/message: defined 'message' as 'hello world'
notice: Finished catalog run in 0.01 seconds
```

同样它会将 notify 资源内容输出到系统标准输出，并显示执行 notify 资源的时间。

4.3.7 puppet module 介绍

puppet module 是 Puppet 的基础模块工具，它包含下载、更新、查找、升级和创建基础模块等功能。我们会经常用到它的查找基础模块功能，它可以从 Puppet Forge 上查找已经开发好的 Puppet 基础模块代码来为我们所用，以减少运维工程师的重复劳动，并提升工作效率。

1. puppet module 常用参数

```
1 --generate
2 --install
3 --list
4 --search
5 --uninstall
6 --upgrade
```

下面简单分析一下 puppet module 工具的常用参数。

- ❑ generate 参数：创建标准的基础模块目录结构。
- ❑ install 参数：安装 Puppet Forge 的基础模块。
- ❑ list 参数：以树形结构显示现有的基础模块（结果输出类似 tree 命令）。
- ❑ search 参数：在 Puppet Forge 中查找基础模块。
- ❑ uninstall 参数：删除已经安装的基础模块。
- ❑ upgrade 参数：升级已经安装的基础模块。

2. puppet module 案例

案例 1

首先通过 search 参数查找在 Puppet Forge 中的 apache 相关基础模块，具体命令如下：

```
# puppet module search apache
Searching http: //forge.puppetlabs.com ...
NAME                          DESCRIPTION              AUTHOR          KEYWORDS
puppetlabs-apache              This is a generic ...  @puppetlabs      apache web
puppetlabs-passenger           Module to manage P...  @puppetlabs      apache
DavidSchmitt-apache            Manages apache, mo...  @DavidSchmitt    apache
jamtur01-httpauth              Puppet HTTP Authen...  @jamtur01        apache
```

根据查找信息，通过 install 参数可以安装 puppetlabs-apache 基础模块，并通过 version 参数指定版本信息，具体命令如下：

```
# puppet module install puppetlabs-apache --version 0.0.2
```

最后，如果觉得安装的版本不是自己想要的版本，还可以卸载它，通过 uninstall 参数卸载 puppetlabs-apache 基础模块，具体命令如下：

```
# puppet module uninstall puppetlabs-apache
```

安装好的基础模块会根据 Master 的主配置文件 puppet.conf 中的 modulepath 参数将基础模块放到指定的目录中，默认路径为 /etc/puppet/modules。

除了通过 puppet module 工具来获取基础模块外，还可以自己开发 Puppet 的基础模块，在第 5 章将会介绍如何自己开发 Puppet 的基础模块。

案例 2

通过 generate 参数创建基础模块。具体命令如下：

```
# puppet module generate example-mymodule
Generating module at /etc/puppet/modules/example-mymodule
example-mymodule
example-mymodule/Modulefile
```

```
example-mymodule/manifests
example-mymodule/manifests/init.pp
example-mymodule/README
example-mymodule/tests
example-mymodule/tests/init.pp
example-mymodule/spec
example-mymodule/spec/spec_helper.rb
```

通过 list 参数查看现有基础模块与版本。具体命令如下：

```
# puppet module list
/etc/puppet/modules
├── examplecorp-apache4 (v1.0)
├── puppetlabs-stdlib (v4.1.0)
```

4.3.8 puppet resource 介绍

puppet resource 是资源抽象层的 shell，通过它可以将当前系统状态转换为 Puppet 的代码，并且它还具有将当前的系统状态改变为 Puppet RAL 状态等功能。下面来看一下 puppet resource 的常用参数和案例。

1. puppet resource 常用参数

以下为 puppet resource 的常用参数，puppet resource 命令由 Puppet 的作者 Luke Kanies 提供。

```
1  --debug
2  --edit
3  --host
4  --param
5  --types
6  --verbose
```

下面简单分析一下 puppet resource 命令的常用参数。

- debug 参数：打开调试信息开关。
- edit 参数：将查询结果写入文件中，在编辑器中打开这一文件，并且以 Puppet 代码的表现形式复述这一文件。
- host 参数：指定之后，连接到已命名主机的资源服务器，获取指定类型资源的列表。
- param 参数：添加更多参数以进行查询输出。
- types 参数：列出所有可获得的类型。
- verbose 参数：输出扩展信息。

2. puppet resource 案例

下面通过 puppet resource 命令来修改系统 root 账号的命令解析器，将 bash 解析器改为

sh 解析器。这里分为 3 个步骤。

步骤 1　通过 puppet resource 命令，后面接 user 参数和 root 系统账户名，将系统 root 账号转换为 Puppet 的代码。

```
# puppet resource user root
```

输出结果如下：

```
user { 'root':
  ensure => 'present',
  gid    => '100',
  groups => ['root'],
  home   => '/home/root',
  shell  => '/bin/bash',
  uid    => '0',
}
```

步骤 2　puppet resource 命令输出后接管道和 tee 命令，管道的作用是将前者的输出变为后者的输入，tee 命令的作用是将结果输出到屏幕，同时导入 change.pp 文件。编辑 change.pp 文件，将 root 账号的 /bin/bash 改为 /bin/sh 命令解析器。

```
# puppet resource user root |  tee change.pp
user { 'root':
  ensure => 'present',
  gid    => '100',
  groups => ['root'],
  home   => '/home/root',
  shell  => '/bin/bash',  # 编辑 change.pp 文件，将 /bin/bash 替换为 /bin/sh
  uid    => '0',
}
```

步骤 3　通过 puppet apply 命令来应用 change.pp 文件中的内容。

```
# puppet apply change.pp
notice: /Stage[main]//User[root]/shell: shell changed '/bin/bash' to '/bin/sh'
notice: Finished catalog run in 0.03 seconds
```

这时 Puppet 会将 root 系统账号的 /bin/bash 改为 /bin/sh 命令解析器，至此整个流程结束。

4.3.9　puppet describe 介绍

puppet describe 是 Puppet 资源帮助文档工具，通过它可以显示资源的使用方法、格式和案例。资源是 Puppet 的核心，如果还在为资源的使用方式发愁，那么赶紧来试试 puppet describe 这个工具。下面来看一下 puppet describe 工具的常用参数和案例。

1. puppet describe 常用参数

以下为 puppet describe 的常用参数，puppet describe 命令由 David Lutterkort 提供。

```
--providers
--list
--meta
--short
```

下面简单分析一下 puppet describe 资源帮助文档的常用参数。

- providers 参数：显示资源对不同平台资源的支持情况。
- list 参数：显示资源列表。
- meta 参数：显示所有 metaparameters（即元参数），在第 7 章会详细介绍。
- short 参数：简明扼要地显示参数信息。

2. puppet describe 案例

如果我们对 file 资源的使用及与它相关的属性都不太了解，可以通过 puppet describe 命令来查看 file 资源的详细参数和使用案例。具体如下：

```
# puppet describe file
Parameters
----------
- **backup**
    Whether files should be backed up before
    being replaced.  The preferred method of backing files up is via
...
```

由于篇幅原因，我们省略了一些输出内容，通过 puppet describe 工具输出了 file 资源的属性介绍，而且还包含了 file 资源的使用案例。

另外，还可以用 -s 表示 short 参数的简写，-m 表示 meta 参数的简写，通过参数的简写方式查看 file 资源的主要信息和元参数等。由于篇幅的原因这里只截取了输出的部分内容。

```
# puppet describe file -s -m
file
====
Manages local files, including setting ownership and
permissions, creation of both files and directories, and
...
```

4.3.10 puppet doc 介绍

puppet doc 是一个将 Puppet 代码中的注释转换为文档的工具。运维工程师在管理配置服务时要将整个服务器配置过程转换为 Puppet 能识别的代码，代码中可以书写配置的注释。那些刚接手配置管理的新人，对很多 Puppet 配置代码是比较生疏的，所以需要一个帮助文档让查询变得更方便。puppet doc 就是这样的一个工具，它可以为 Puppet 语言、节点和类创建帮助文档，从而使那些对系统生疏的人快速了解系统。下面来看一下 puppet doc 的常用参数和案例。

1. puppet doc 常用参数

以下为 puppet doc 的常用参数，此命令由 Puppet 的作者 Luke Kanies 提供。

```
1  --all
2  --outputdir
3  --mode
4  --reference
5 --charset
6 --manifestdir
7 --modulepath
```

下面简单分析一下 puppet doc 命令的常用参数。

- all 参数：在 rdoc 模式下，此参数可以输出详细的文档资源类型。
- outputdir 参数：生成文档的输出路径，此参数只支持 rdoc 模式。
- mode 参数：后接输出模式，目前 doc 支持的输出模式包括 text、pdf 和 rdoc，默认模式是 text 模式。如果使用 rdoc 模式，则需要提供 manifests-path 路径。
- reference 参数：生成特定的文档信息。
- charset 参数：设置字符集，此参数只支持 rdoc 模式。
- manifestdir 参数：设置 Puppet 的 manifests 路径，如果不设置此参数，它会在 puppet.conf 寻找 manifestdir 默认路径。此参数只支持 rdoc 模式。
- modulepath 参数：设置 Puppet 的基础模块路径，如果不设置此参数，它会在 puppet.conf 寻找 modulepath 默认路径。此参数只支持 rdoc 模式。

2. puppet doc 案例

写明注释可以让我们在查询历史代码时更快地了解当时代码编写的意图，同时也方便他人更快熟悉 Puppet 配置管理系统状态。例如，要将 /etc/puppet/manifests/puppet.pp 文件内容生成文档，可在代码中通过 # 来增加注释，具体如下：

```
class puppet {
    # This class sets up the Puppet client.
    #
    # ==Actions
    # Install a cron job to run Puppet.
    #
    # ==Requires
    # * Package["puppet"]
    #
    cron { "run-puppet":
      command => "/usr/sbin/puppet agent --test >/dev/null 2>&1",
      minute => inline_template("<%= hostname.hash.abs % 60 %>"),
    }
}
```

通过 puppet doc 命令来生成 Puppet 代码的 HTML 手册，puppet doc 后接参数 all 表示输出详细的文档资源类型，ouputdir 表示 HTML 输出路径，mode 表示生成模式，manifestdir 后接输入 Puppet 代码的目录或路径，具体如下：

```
# puppet doc --all --ouputdir=/var/www/html/puppet -mode rdoc \
--manifestdir=/etc/puppet/manifests/
```

puppet doc 工具会根据需求生成 HTML 文档，如图 4-5 所示。

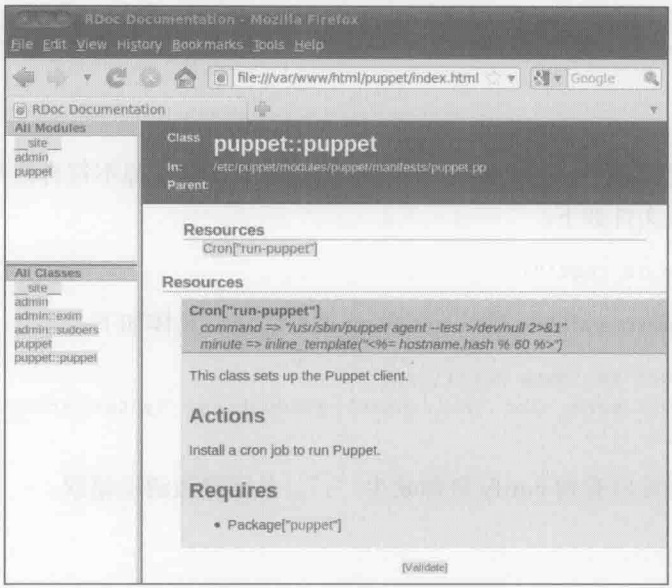


图 4-5　HTML 格式文档

生成文档后，可以方便他人阅读，并了解目前通过 Puppet 配置服务器的状况。


注意　rdoc 是目前比较流行的 Ruby 文档生成器。

4.3.11　puppet parser 介绍

puppet parser 命令主要用于 .pp 文件语法检查。在用 Puppet 配置管理系统时，需要养成一个比较好的习惯，就是当我们编写好 .pp 代码后，在上线前最好通过 puppet parser 命令进行语法检查，通过该项检查后再上线 .pp 文件。检查 .pp 文件语法是否正确需要加 validate 动作，我们来看一下 puppet parse 的应用案例。

编辑 notify.pp 测试文件，命令如下：

```
notify {"this is test!": }
```

notify.pp 文件的作用是调用 notify 资源在屏幕上输出“this is test”。

1. 测试语法

通过 puppet parser validate 检查 notify.pp 文件的语法，如果无返回，则表示 notify.pp 文件正确。(笔者觉得如果语法正确应该显示 Syntax ok，这样用户体验会更好一些，但是目前利用命令检查 .pp 文件后，如果语法正确却不显示任何信息，我们期待 Puppet 能对这一功能进行完善。)

```
# puppet parser validate notify.pp
```

2. 模拟错误

去掉 notify 资源的“:”来模拟错误，这会导致 notify 资源不符合语法规范，最终结果会报错。notify.pp 文件如下：

```
notify {"this is test!"}
```

通过 puppet parser validate 确认 notify.pp 文件语法，具体如下：

```
# puppet parser validate notify.pp
err: Could not parse for environment production: Syntax error at '}' at /etc/
puppet/notify.pp:1
```

通过语法检查可以看到 notify 资源缺少“:”，从而导致语法错误。

3. 批量确认

当我们要检查的 .pp 文件比较多时，还可以通过系统辅助 find 命令来确认 .pp 代码语法是否正确，具体如下：

```
# find /etc/puppet/ -type f -name "*.pp" | xargs -n 1 -t puppet parser validate
```

4.3.12 puppet 帮助命令介绍

puppet help 与 puppet man 均为 Puppet 命令的帮助手册，它们输出的结果基本一致，只不过 puppet help 查看帮助命令会一次性输出所有信息，而 puppet man 要指定查看的命令。

1. puppet man 方式

通过 puppet man 后接命令方式可以查看命令帮助信息。如以下代码可以参看到 agent 的相关文档与参数手册。退出时输入“q”即可。

```
# puppet man agent
puppet-agent(8) -- The puppet agent daemon
========
```

```
SYNOPSIS
--------
Retrieves the client configuration from the puppet master and applies it to
the local host.
This service may be run as a daemon, run periodically using cron (or something
similar), or run interactively for testing purposes.
...
```

由于篇幅的原因这里只输出了部分内容。

2. puppet help 方式

puppet help 后接命令的方式，作用与 puppet man 一样，不过它会一次性输出所有命令的相关帮助信息，所以笔者在后边加了“|”管道，将它的输出导入了 more 命令，以便更方便地看到帮助信息。具体如下：

```
# puppet help agent | more
puppet-agent(8) -- The puppet agent daemon
========
SYNOPSIS
--------
Retrieves the client configuration from the puppet master and applies it to
the local host.
This service may be run as a daemon, run periodically using cron (or something
similar), or run interactively for testing purposes.
USAGE
...
```

由于输出的内容比较多，所以做了截取。由于它们都比较简单，在这里就不深入介绍了。

4.3.13 puppet filebucket 介绍

运维工程师经常要做的就是为服务器备份和恢复数据，Puppet 作为运维工程师的利器也为我们提供了相应的解决方案——filebucket 工具。通过 puppet filebucket 命令可以远程备份或恢复文件，其有 3 种工作模式，即备份、获取和还原。此命令由 Puppet 的作者 Luke Kanies 提供。下面来看一下 puppet filebucket 的工作模式和案例。

1. puppet filebucket 工作模式

1）备份：数据备份是运维工程师日常工作中的重要环节，通过 puppet filebucket backup 参数就可以将一个或多个文件发送到 file bucket 文件桶进行备份，同时它会打印出每个备份文件的 MD5[⊖]值。

2）获取：通过 puppet filebucket get 参数可以获取 Agent 的文件内容。

3）还原：通过“puppet filbucket restore 参数 + 路径 + MD5”的形式可以还原此文件

⊖ MD5（Message Digest Algorithm）为计算机安全领域广泛使用的一种散列函数，用以提供对消息的完整性保护。

的内容。

2. puppet filebucket 案例

以下是 puppet filebucket 的 3 种模式，分别是备份、获取和还原备份文件。通过这 3 种模式来实现线上文件的备份与恢复。具体内容如下：

1）备份线上文件，backup 参数后接备份文件路径和文件名称。

```
# puppet filebucket backup /etc/passwd
```

2）获取备份的文件，get 参数后接备份原始文件的 md5sum 值。

```
# puppet filebucket get a0beabf0f7e6b234f26482247b147ff9
```

3）恢复备份的文件，restore 参数后接恢复路径、文件名和备份文件 md5sum 值。

```
# puppet filebucket restore /tmp/passwd  a0beabf0f7e6b234f26482247b147ff9
```

4.3.14 puppet file 介绍

puppet file 命令主要用于检索和存储 filebucket 中的内容。下面介绍如何利用文件名和 MD5 方式从 Master 的 filebucket 中下载文件。

文件名方式如下：

```
# puppet file download puppet:///modules/editors/vim/.vimrc
```

MD5 方式如下：

```
# puppet file download {md5}8f798d4e754db0ac89186bbaeaf0af18
```

其实在实际工作中通过 puppet file 下载文件的方式使用得并不多，它只解决临时的一些下载需求，如果要传的数据比较多或文件比较大，通常通过 scp 或 rsync 的方式来同步 Agent 与 Master 之间的文件。这里推荐使用 rsync 或通过 exec 资源调用 rsync 的方式来同步文件，因为 rsync 是一个比较专业的数据同步软件。

4.3.15 puppet kick 介绍

puppet kick 命令是一个远程 Agent 管理控制工具，通过它可以主动要求某台服务器更新配置，而且它支持查看 Agent 存活状态。下面来介绍一下 puppet kick 的常用参数和案例。

1. puppet kick 常用参数

puppet kick 的常用参数由 Puppet 的作者 Luke Kanies 提供。具体如下：

```
1  --all
2  --class
```

```
3  --debug
4  --foreground
5  --host
6  --ignoreschedules
7  --parallel
8  --tag
9  --ping
10 --test
```

下面简单分析一下 puppet kick 命令的常用参数。

- all 参数：连接所有 Agent（需要 LDAP 支持）。
- class 参数：连接一类 Agent（需要 LDAP 支持）。
- debug 参数：打开调试信息开关。
- foreground 参数：在前台运行每一个配置，当连接到一个 Agent 时，只有 Agent 端更新完毕才会退出继续执行下一个，默认是关闭的。
- host 参数：可以指定 Agent，这个可以出现多次，例如 host x.xx.com 和 host y.xx.com。
- ignoreschedules 参数：决定客户端在运行配置的时候是否要忽视过程。这一功能可以用来强制客户端执行其原本不会执行的特别迅速的任务。默认设置是关闭的。
- parallel 参数：并行指示每一个客户端应该连接哪一个进程，参数默认是 1，也就是按照顺序依次连接每个客户端。
- tag 参数：指定一个 tag（标志）来选择应用的对象。不可以与 test 参数一同应用。
- ping 参数：发送 ICMP 包到指定的客户端，并忽略那些没有回应的客户端。
- test 参数：打印可以连接的客户端，但并不会真正连接它（需要 LDAP 支持）。

2. puppet kick 案例

puppet kick 是一个远程 Agent 管理控制工具，通过它可以主动连接 Agent 的进程，并要求这些被连接的机器主动更新配置，最常见的使用方式就是 Master 访问 Agent 或 tag 并要求这些 Agent 或 tag 的对象都主动运行一次。下面以 /etc/puppet/modules/test/manifests/init.pp 文件为例，让 Agent 主动更新 tag 为 test 的资源。以下为具体代码：

```
$content = "some content"
file {'/tmp/testing':
  ensure => file,
  content => $content,
    tag => 'test'
}
```

通过 puppet kick 命令主动更新 Agent 状态前，需要确定 Agent 是否已经监听了 TCP 8139 端口，可以修改 /etc/puppet/puppet.conf 文件，追加如下内容或设置 --listen 参数。

```
#/etc/puppet/puppet.conf
```

```
[agent]
listen = true
```

再次确认名称空间是否对 puppetrunner 授权，编辑 /etc/puppet/namespaceauth.conf，追加以下内容。

```
[puppetrunner]
allow example.puppet.com
```


注意 在 Agent 启动监听 TCP 8139 端口后，还要确认 iptables 是否已经对 TCP 8139 端口放开限制。解除 TCP 8139 端口限制命令：iptables –t filter –A INPUT –p tcp –m state –state NEW --dport 8139 –j ACCEPT。

确认启动 8139 端口后还要设置 Agent 的访问路径，修改 /etc/auth.conf 文件，并准许访问 /run 路径。在 Puppet 2.7 版本中，Agent 把代码加到了 auth.conf 文件中。

```
#/etc/puppet/auth.conf
path /run
method find, search, save
auth yes
allow example.puppet.com
```

确认 Agent 监听 TCP 8139 端口和授权 /run 访问路径后便可以使用，通过 puppet kick 命令发送 ICMP 包给指定的 Agent，并忽略那些没有响应的机器。具体如下：

```
# puppet kick -p 10 -t test -host example.puppet.com
```

代码执行后可以看一下 Agent 端的 /tmp/testing 内容与时间戳是否已经变更。

第 5 章 Chapter 5

通过 Puppet 构建主机

目前我们已经对 Puppet 的安装环境、配置文件和常用命令参数有了基本的了解。本章主要串联前几章节的知识点来讲解 Agent 首次访问 Master 的配置过程，介绍 Puppet 配置管理的两个重要组成部分——manifests 和 modules 目录，包括两个目录都存放了什么，作用是什么，并结合 Apache 案例介绍如何通过 Puppet 构建主机。另外当我们通过 Puppet 来管理海量的服务器时，错误的配置可能导致严重的后果，Puppet 是如何解决这样问题的呢？下面让我们带着这些问题来寻找答案。

5.1 Agent 首次访问 Master 配置过程

Puppet 配置管理工具的常见使用场景就是 Agent 访问 Master 来获取配置信息，也就是我们之前介绍的 C/S 架构，这里让我们以最基本的方式来配置 Master 和 Agent，看它们是如何工作的。首次配置 Agent 访问 Master，笔者将它分为 4 个部分，每个部分又有详细的步骤和注意事项。首先创建 site.pp 文件和目录，然后启动 Master 守护进程，再次确认防火墙和网络配置，最后通过 Agent 访问 Master，整个过程结束。下面让我们来分别看一下这 4 部分内容。

5.1.1 创建 site.pp 文件和目录

/etc/puppet/manifests/site.pp 文件（下称 site.pp）是 Puppet 站点的导航文件，Agent 访问 Master 的一切配置管理工作都由 site.pp 文件开始的，它的作用是告诉 Master 寻找并载入 Agent 的配置信息。默认情况下 site.pp 文件会存放在 /etc/puppet/manifests 目录中，我们要

提前确认一下 /etc/puppet/manifests 目录和 site.pp 文件是否存在，如果它们不存在则 Master 是拒绝启动的。所以首次配置 Puppet 的话，应先自行创建上述目录和文件。创建命令如下：

```
# mkdir /etc/puppet/manifests
# touch /etc/puppet/mainifests/site.pp
```

创建 manifests 目录和 site.pp 文件，为了后续调试 Agent 访问 Master 过程中方便看到结果。首次可以将 notify 资源追加到 site.pp 文件中，并在 notify 资源内赋值“hello world”，notify 资源的作用就是将值输出到屏幕上。具体如下：

```
# echo 'notify{"hello world": }' > /etc/puppet/mainifests/site.pp
```

当 Agent 访问 Master 时会调用 site.pp 文件中的 notify 资源，将其值（hello world）在屏幕上输出。如果我们看到了屏幕的输出，这就表明 Agent 已经成功地访问了 Master。

注意 manifests 是 Puppet 中的术语，是指包含配置信息的目录。Puppet 所有配置文件都以 .pp 作为扩展名。manifests 目录和 site.pp 文件的默认路径可以在 puppet.conf 的 [master] 区段中修改，通过修改 puppet.conf 中的 mainfestdir 参数来更新 manifests 默认值，通过修改 manifest 参数来修改 site.pp 文件默认值。

5.1.2 Master 配置

Master 在 UNIX/Linux 系列服务器上以 Demon（中文翻译为守护进程，下称守护进程）形式启动，启动后默认监听 TCP 8140 端口向外提供的服务。在启动 Master 前首先确认 puppet.conf 配置文件是否已经存在并正确配置，这里可以参考第 4 章中的 puppet.conf 来配置。确认 puppet.conf 配置文件没有问题后，通过以下命令和参数方式启动 Master 的守护进程，其中 daemonize 参数表示将进程信息发送到标准输出；verbose 参数表示输出扩展信息，我们将它的输出信息重定向到 master.log 文件中。实现命令如下：

```
# nohup puppet master --verbose --no-daemonize >> master.log 2>&1 &
```

启动 Master 前，守护进程会对一些目录、文件权限、主要配置文件进行预检，预检内容包括相关文件 / 目录是否存在，各目录和文件权限是否正确等。通过这种启动方式默认会输出详细的预检的信息到屏幕上（注：由于我们重定向了输出日志，这里可以到 master.log 文件中查看 Master 输出信息），如图 5-1 所示。若守护进程首次启动失败，这些输出的预检信息可以方便我们定位和查找失败的原因。

Master 守护进程正常启动后，最好再通过系统命令 netstat --tnl 确认 TCP 8140 端口是否启动，如图 5-2 所示。如果端口启动则说明 Master 已经正常工作。

```
notice: Starting Puppet master version 2.7.21
info: mount[files]: allowing * access

info: access[^/catalog/([^/]+)$]: allowing 'method' find
info: access[^/catalog/([^/]+)$]: allowing $1 access
info: access[^/node/([^/]+)$]: allowing 'method' find
info: access[^/node/([^/]+)$]: allowing $1 access
info: access[/certificate_revocation_list/ca]: allowing 'method' find
info: access[/certificate_revocation_list/ca]: allowing * access
info: access[^/report/([^/]+)$]: allowing 'method' save
info: access[^/report/([^/]+)$]: allowing $1 access
info: access[/file]: allowing * access
info: access[/certificate/ca]: adding authentication any
info: access[/certificate/ca]: allowing 'method' find
info: access[/certificate/ca]: allowing * access
info: access[/certificate/]: adding authentication any
info: access[/certificate/]: allowing 'method' find
info: access[/certificate/]: allowing * access
info: access[/certificate_request]: adding authentication any
info: access[/certificate_request]: allowing 'method' find
info: access[/certificate_request]: allowing 'method' save
info: access[/certificate_request]: allowing * access
info: access[/]: adding authentication any
info: access[/tmp]: adding authentication any
info: access[/tmp]: allowing 'method' find
info: access[/tmp]: allowing 'method' save
info: access[/tmp]: allowing * access
info: Inserting default '/status' (auth true) ACL because none were found in '/etc/puppet/auth.conf'
```

图 5-1 启动守护进程

```
[redacted]:/etc/puppet # netstat -tnl | grep 8140
tcp        0      0 0.0.0.0:8140            0.0.0.0:*               LISTEN
```

图 5-2 确认 Master 启动方式

注意 Master 的守护进程启动后默认会将日志信息输出到系统的 Syslog，这些信息包含 Agent 访问信息和从 Master 拉取的配置信息等。读者可以根据安装系统的发行版本来查找守护进程的输出信息，RedHat 和 Suse 可以查看 /var/log/messages，Debian 和 Ubuntu 可以到 /var/log/daemon.log 中查找输出信息。

Master 启动后会根据 puppet.conf 文件中的 ssldir 参数的路径创建证书目录，用来缓存 SSL 信息和证书文件，如图 5-3 所示。

```
drwxrwx--x  8 puppet users  4096 Oct  8 15:19 .
drwxrwxrwx 12   1004 users  4096 Jan 20 18:59 ..
drwxrwx---  5 puppet puppet 4096 Dec 17 16:40 ca
drwxr-xr-x  2 puppet users  4096 Oct 22 15:31 certificate_requests
drwxr-xr-x  2 puppet users  4096 Oct 22 15:31 certs
-rw-r--r--  1 puppet users   922 Oct  8 15:19 crl.pem
drwxr-x---  2 puppet users  4096 Oct  8 15:19 private
drwxr-x---  2 puppet users  4096 Oct 22 15:31 private_keys
drwxr-xr-x  2 puppet users  4096 Oct 22 15:31 public_keys
```

图 5-3 证书目录

5.1.3 防火墙配置

确认 Master 守护进程成功启动后，还需要确认 Master 所在机器的 iptables（防火

墙）[⊖]是否已经开放本机 TCP 8140 端口访问限制，以便 Agent 可以正常访问 Master 而不受规则限制。

开放 iptables 防火墙对 8140 端口的限制。具体如下：

```
# iptables -t filter -A  INPUT -p tcp -m state -state NEW --dport 8140 -j ACCEPT
```

下面来了解一下 iptables 的参数。

- -t 为参数接表名，目前它有 3 个表，即 mangle 表、nat 表和 fileter 表。其中 mangle 表用来对数据打标记，nat 表对数据进行转发，filter 表主要用来过滤数据。
- -A 为参数接链名，不同的表中的链是不一样的，目前 filter 表有 3 个链，即 INPUT 链、OUTPUT 链和 FORWARD 链。其中 INPUT 链表示进入防火墙的数据，OUTPUT 链表示离开防火墙数据，FORWARD 链为通过 nat 表转发的数据。
- -p 参数后接协议，目前支持 3 种协议，即 tcp、udp 和 icmp 协议。
- -m 参数后接 stat，其拥有 4 种状态，即 INVALID、ESTABLISHED、NEW 和 RELATED，其中 NEW 状态表示将要或正在建立的第一个连接。
- --dport 参数后接目地端口。
- -j 参数后接 ACCEPT（其含义为放行）或 DROP（其含义为丢弃）。

注意 根据所在的网络环境，成功启动 Master 的 TCP 8140 端口后，除了加防火墙规则外还需要确认网络层面和路由器的策略是否会影响到 Master 和 Agent 的正常访问。iptables 命令的参数区分大小写。

5.1.4 Agent 配置

相对于 Master 的配置来说，Agent 的配置在这里要更简单一些，建议初学者将 Agent 独立运行于 crontab 定时任务中，只需要告诉它 Master 的域名和 IP 地址就可以了。首先需要在 Agent 上配置指向 Master 的域名，这里有两种配置方式：一种方式是在第 3 章介绍过的 Dnsmasq 系统，作为一个轻量级的域名解析系统，在维护 1000 台以下服务器时可以使用它；另一种更简单一些，这里推荐测试时通过设置 /etc/host 文件的形式来指定 Master 的域名。笔者将整个 Agent 访问 Master 过程分为了 4 个步骤。

步骤 1　在 Agent 上设置 host。

首次访问需要在 Agent 上配置指向 Master 的域名，但是域名需要通过 DNS（域名系统）注册后才能使用，而注册域名并不是我们学习的重点，所以为了降低学习的成本，测试期间最简单的方法就手动编辑 /etc/hosts 文件，在其中配置一个虚拟域名（即在 DNS 系统中

⊖ 关于 Iptables 的更多信息可以参考互联网上《Iptables 指南 1.1.19》这篇文章。

没有注册的域名），系统会优先解析 DNS 中的域名，当找不到要解析的域名后，访问本机 /etc/hosts 文件继续解析域名的关系，而我们可以通过在 /etc/hosts 增加虚拟域名访问关系的方式来绕开 DNS，以满足 Master 访问 Agent 的环境需求。增加虚拟域名（puppet.example.com）到 /etc/hosts 文件方式具体命令如下：

```
# echo "puppet.example.com  192.168.1.1" >> /etc/hosts
```

追加虚拟域名后要通过系统 ping 命令确认域名是否可以正常解析。

```
# ping puppet.example.com
```

步骤 2　测试 Agent 访问 Master。

确认配置的虚拟域名可以正常解析后，在 Agent 通过 puppet agent 命令后接 server 参数加域名（如 puppet.example.com）指定 Master 服务器的域名；加 test 参数，test 参数是一些命令的集合，通过它可以打开 onetime、verbose、ignorecache、no-daemonize、no-usecacheonfailure、detailed-exit-codes、no-splay 和 show diff 等一系列参数。若首次访问会提示（peer certificate wont be verified in this SSL session）认证失败信息，具体如下：

```
# puppet agent --server puppet.example.com --test
info: Creating a new SSL key for puppet_agent
warning: peer certificate won't be verified in this SSL session
warning: peer certificate won't be verified in this SSL session
notice: Did not recive certificate
```

之所以会出现上边的认证失败，是因为 Master 没有对 Agent 授权签名，这时需要到 Master 上对 Agent 的 Hostname 授权签名。


注意　如果不想在 puppet agent 参数后指定 Master 的域名，也可以在 Agent 的配置文件 /etc/puppet/puppet.conf 的 [main] 段增加参数 server=puppet.example.com，增加后可以直接在终端通过 puppet agent --test 的形式访问 Master。

步骤 3　Master 对 Agent 进行签名。

如步骤 2，当 Agent 首次访问 SSL 信息认证失败后，Master 需要通过 puppet cert 命令加 list 参数的方式查看等待证书认证的机器列表，具体如下：

```
# puppet cert --list
" puppet_agent " (8F: 40: 14: 96: 7C: 29: 44: 01: FC: 33: EE: 84: 58: 9C: B2: 71)
```

其中 puppet_agent 是等待签名 Agent 的 Hostname。再次通过 puppet cert 加 sign 参数加 Agent 的 Hostname（如 puppet_agent）的方式授权证书签名，具体如下：

```
# puppet cert --sign puppet_agent
Signed puppet_agent
```


提示 在第4章中我们介绍过除了在Master上手动签名外，还可以通过autosign.conf配置文件对Agent进行自动签名。如果读者对自己的网络策略与安全状况比较了解，推荐通过autosign.conf自动授权方式来授权签名。

步骤4 Agent成功访问Master并获取配置信息。

在Master对Agent的证书签名后，再次返回Agent机器，这时Master已经授权Agent的证书签名，签名后的Agent根据Hostname信息匹配site.pp文件中的配置信息并显示响应结果，具体如下：

```
# puppet agent --server puppet.example.com --test
info: Creating a new SSL key for puppet_agent
warning: peer certificate won't be verified in this SSL session
info: Caching certificate for ca
warning: peer certificate won't be verified in this SSL session
warning: peer certificate won't be verified in this SSL session
info: Creating a new SSL certificate request for puppet_agent
info: Certificate Request fingerprint (md5): 8F: 40: 14: 96: 7C: 29: 44: 01: FC: 33:
EE: 84: 58: 9C: B2: 71
notice: Starting Puppet client version 2.7.25
info: Caching catalog for puppet_agent
info: Applying configuration version '1375433624'
notice: hello world    # notify资源输出信息
notice: /Stage[main]//Node[default]/Notify[default]/message: defined 'message' as
'hello world'
notice: Finished catalog run in 0.15 seconds
```

从以上显示的结果中可以看到，5.1.1节中的site.pp文件，通过调用系统notify资源将“hello world”信息输出到Agent标准输出，说明Agent已经成功获取Master的配置信息。到目前为止我们已经成功完成了配置一台Agent服务器访问Master的过程，并从Master获取相应的信息。整个配置过程是比较简单的，笔者觉得容易出问题的是证书认证环节。在这个环节可能导致证书出问题的情况有很多种，所以没有在这里过多介绍，需要读者根据自己的情况通过搜索引擎来寻找答案。更多信息请参考官方网站http：//projects.puppetlabs.com/projects/puppet/wiki/ruby_ssl_2007_006。

注意 多数连接失败可能是系统时间不统一导致的，SSL连接依赖主机上的系统时间，如果Master和Agent上的系统时间不正确，很有可能导致连接的失败或者得到错误信息导致证书不被信任。读者可以通过NTP（网络时间协议）来尽量确保机器时间的正确和统一。

5.2 manifests和modules目录介绍

在上一节我们介绍了Agent首次访问Master的配置步骤和注意事项，其只是通过Puppet

来管理海量服务器的一个开始，我们还需要继续深入了解 manifests 和 modules 这两个目录，它们才是 Puppet 配置管理服务器的重要组成部分。其中 manifests 目录用于存放服务器配置管理文件，这些文件需要以 .pp 为文件扩展名（建议文件使用 UTF-8 编码字符集，为后续与 Puppetdb 结合奠定基础）。另外我们也可以将 manifests 目录看做配置管理的清单目录，清单中包含代码逻辑和 Agent 入口文件。逻辑部分会在第 6 章详细介绍，本章主要介绍 Agent 的入口文件，即 site.pp 文件。modules 目录又称基础模块目录，可以将它看做仓库，仓库中的模块可以提供清单文件重复使用，仓库中主要存放 class 类文件和基础模块相关的配置文件等。

5.2.1 manifests 目录介绍

manifests 目录中存放了配置管理代码逻辑和 site.pp 入口文件，所有的 Agent 访问 Master 时都优先匹配到 site.pp 文件中的 node 节点，节点与节点间有支持继承。这里又多了两个概念 node 节点与继承，那什么是 node 节点？什么又是继承？它的好处是什么呢？下面先来介绍一下 node 节点和 node 节点的使用案例，然后介绍继承的含义和案例。

1. node 节点

在之前的章节中我们将 Puppet 的客户端称为 Agent，在本小节中 Agent 在 site.pp 文件中又被称为 node（中文翻译节点，下称 node 节点）。其实它们是一个意思，只是一个标识 Master 端，一个标识 Agent 端。当 Agent 访问 Master 获取配置时，Master 会通过 facter 工具获取 Agent 的 Hostname，并通过 Hostname 自动匹配 site.pp 文件中的 node 节点拉取配置信息，这就是 node 节点的作用。site.pp 文件中可以设置一个 node 节点，也可以设置多个，如图 5-4 所示。它既支持正则表达式匹配，也支持 node 节点间继承，node 节点功能灵活而又强大。

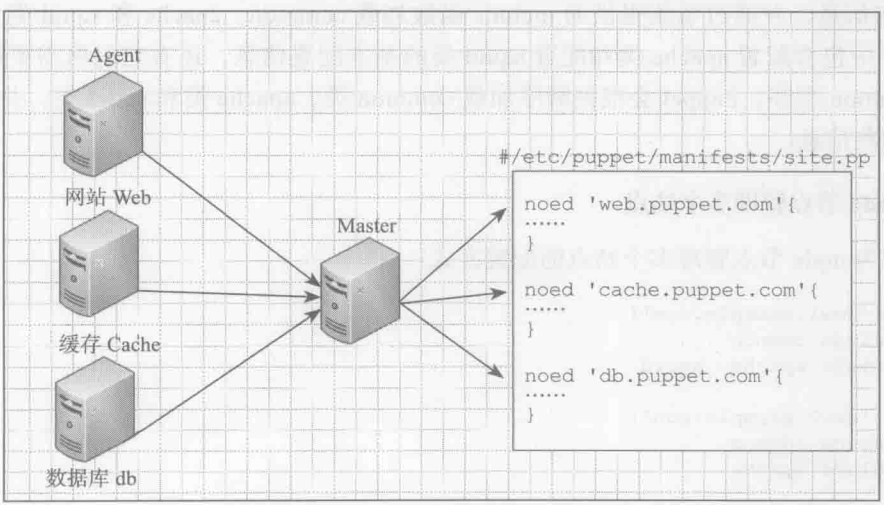


图 5-4 多 Agent 访问 Master 的 node 节点

下面来看一下 site.pp 文件中定义的 node 节点格式。

```
node 'web.puppet.com' {
  # 省略
}
node 'cache.puppet.com' {
  # 省略
}
node 'db.puppet.com' {
  # 省略
}
```

在 site.pp 文件中通过 node 节点分别定义 web.puppet.com、cache.puppet.com 和 db.puppet.com 来管理不同来源的 Agent 节点，当 Agent 访问 Master 时就会匹配这些节点，并获取 node 节点中的配置信息。下面介绍一下通过 node 节点来分别管理一个站点、多个站点、正则表达式匹配站点和默认方式管理站点的使用场景（由于篇幅原因，本小节只以代码片段形式介绍 node 节点在各种情况下的使用场景）。

2. node 节点管理一个站点

以下为 node 节点管理一个站点的配置方式：

```
node 'www1.example.com' {
  include common
  include apache
  include squid
}
```

其中 node 为系统关键字，ww1.example.com 是 Agent 的 Hostname；大括号中内容为基础模块信息。当 Agent 访问 Master 时会根据它的机器 Hostname 自动匹配 site.pp 文件中 node 节点信息，并通过系统提供的 include 函数加载 common、apache 和 squid 类文件，这些类文件中包含配置 apache 类和配置 squid 类的基本配置信息，还有它们所需要用到的公用库 common 类等，Puppet 会根据顺序加载 common 类、apache 类和 squid 类，并在 Agent 上应用这些信息。

3. node 节点管理多个站点

以下为 node 节点管理多个站点的配置方式：

```
node 'www1.example.com'{
  include common
  include apache, squid
}
node 'www2.example.com'{
  include common
  include apache
}
```

www1.example.com 和 www2.example.com 分别代表不同组 Agent 的 Hostname，当 Hostname 为 www1.example.com 时，访问 Master 会依次加载 common 类、apache 类和 squid 类文件；当 Hostname 为 www2.example.com 时会加载 common 类文件和 apache 类文件；如果多个 Agent 访问 Master，且需要加载的类文件是一致的，也可以用以下的书写格式：

```
node 'www1.example.com','www2.example.com','www3.example.com' {
  include common
  include apache, squid
}
```

这些站点访问 Master 时会依次加载 common 类、apache 类和 squid 类文件，并拉取这些类中的配置信息。

4. node 节点正则表达式方式管理站点

如果管理的站点比较多而且有一定的规律，还可通过正则表达式来管理它们。在 site.pp 文件中支持使用 Ruby 的正则表达式，通过正则表达式可以让我们方便地匹配有规律的 node 节点。下面来匹配一个以 www 开始后接一个或多个数字开头的网站，如 www1、www2，直到 www10，通过以下正则表达式都可以匹配到。

```
node /^www\d+$/ {
  include common
}
```

当 Agent 的 Hostname 来自 www1.example.com、www2.example.com 直到 www10.example.com 时，都会匹配 site.pp 中的这个正则表达式，并加载 common 类文件。

我们再来看一个例子，匹配来源是 foo.example.com 或 bar.example.com 两个 Agent 站点信息，正则表达式如下：

```
node /^(foo|bar)\.example\.com$/ {
  include common}
```

通过正则表达匹配到 foo.example.com 或 bar.example.com 两个 Agent 时会加载 common 类文件。

5. 默认 node 节点

Agent 访问 Master 时会读取 site.pp 文件中的 node 节点列表，并从上到下依次匹配 node 节点中的配置信息，如果没有匹配到相应的 node 节点它就会报错，并提示 Hostname 的配置节点不存在。这种情况多数可能是由于新增 Agent 导致的错误，但对刚接触 Puppet 的用户来说信息并不友好，不知道下面该做什么。有什么办法可以避免这种报错吗？其实不用担心，还可以在 site.pp 文件中设置 default 默认节点。当 Agent 访问 Master 没有匹配到 node 节点信息时，最后就会匹配到 default 默认节点并加载信息。具体的设置方法如下：

```
node default {
  notify{
   "error! not match your node , this is default node":
  }
}
```

当 Agent 访问 Master 的 site.pp 文件没有匹配到相应的 node 节点时，就会匹配这个默认节点，默认节点中会调用 notify 资源在屏幕输出“ error! not match your node ，this is default node”，这样的处理方式也形成 Puppet 配置管理的一个闭环，提升异常情况下的用户体验。

5.2.2 modules 目录介绍

modules 目录又称“基础模块”目录，它由不同的目录和 class 类文件组成，这些目录和文件是完成一个任务的子集，最终可以通过 manifests 串联 modules 中的这些子集来完成一个完整任务，这就是基础模块的作用。获取 modules 基础模块的方式有两种，一种是从官方获取基础模块，另一种是自己开发基础模块。首先来了解一下 modules 基础模块的目录结构，然后了解如何从官网获取基础模块。

1. modules 基础模块目录结构

标准的 modules 基础模块目录包含 6 个目录，分别是 manifests、files、templates、lib、tests 和 spec。来看一下这些目录的作用和目录内存放的文件都是什么。

以下是 modules 基础模块的目录格式。

```
1 /etc/puppet/modules/my_module
2                         |_manifests/
3                         |_files/
4                         |_lib/
5                         |_templates/
6                         |_tests/
7                         |_spec/
```

下面简单分析一下 Puppet 基础模块的目录中存放的内容。

- 第 1 行：基础模块默认根路径，其中 my_module 为基础模块的名字。
- 第 2 行：manifests 目录存放基础模块的类文件和资源等。
- 第 3 行：files 目录存放基础模块的配置文件，目录中配置文件可以通过 Puppet 文件协议下载到 Agent 上。
- 第 4 行：lib 目录存放 Puppet 的插件、自定义函数、Provider 和库文件等，更多信息请参考官方网站 http://docs.puppetlabs.com/guides/plugins_in_modules.html。
- 第 5 行：templates 目录存放 ERB 模板文件，会在第 8 章详细介绍 Puppet 的 ERB 模板文件。

- 第 6 行：tests 目录存放 Puppet 基础模块类测试的用例文件。
- 第 7 行：spec 目录存放测试文件和插件库文件。

注意 第 2 行 modules 基础模块中的 manifests 目录经常与 /etc/puppet/manifests 目录混淆，这两个目录并不一样，需要读者注意。

2. Puppet Forge 获取基础模块

Puppet Forge 是一个免费的 modules 基础模块仓库，我们可以无需独立开发基础模块，直接从官方网站上获取我们想要的基础模块。另外 Puppet Forge 也提供途径让发布自己开发的模块，很多热心的网友将自己开发好的 modules 基础模块通过官方网站分享给他人使用。目前 Puppet Forge 提供两种方式获取基础模块，一种方式可以在官方网页搜索基础模块，另一种方式是通过 Puppet 提供的 puppet module 工具命令来搜索和安装基础模块。如果读者所在的环境可以访问网络，建议通过 puppet module 工具命令方式来安装基础模块；如果不能访问网络，可以通过能访问网络的机器到官网搜索基础模块，下载后将基础模块包移动到生产环境中的 modules 路径上。下面来了解一下通过网站获取基础模块和通过 puppet modules 工具获取基础模块的两种方式。

（1）通过网站搜索

通过官方网站搜索基础模块可以访问 http：//forge.puppetlabs.com/ 网站，借助网站的右边搜索框可以搜索到需要的基础模块，如图 5-5 所示。将搜索到的基础模块下载后放到 Master 的基础模块默认路径（/etc/puppet/modules/）就可以了。

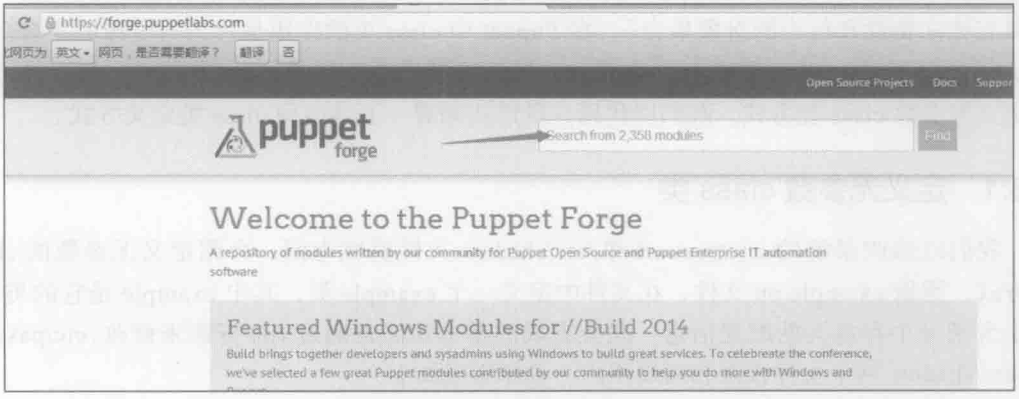


图 5-5 官网搜索基础模块

（2）puppet module 工具获取模块

另一种获取基础模块的方式在第 4 章介绍过，就是通过 puppet module 工具来获取基础模块。puppet module 工具可以通过 install 参数安装基础模块，通过 search 参数查找相应的

基础模块，具体如下。

通过 search 参数查找 Apache 的基础模块如下：

```
# puppet module search apache
```

通过 install 参数安装 Apache 的基础模块，并通过子参数 --version 指定 Apache 的版本。除了 version 子参数外，还包含以下的辅助参数。

- --force 参数：强制重新安装模块。
- --environment 参数：指定安装环境。
- --modulepath 参数：指定模块目录，通常用于指定自定义的模块目录。
- --igore-dependencies 参数：忽略依赖。

```
# puppet module install puppetlabs-apache --version 0.0.2
```

puppet module 工具会从 puppetlabs 官方网站下载 Apache 的基础模块，并将基础模块安装到系统模块的默认路径，即 /etc/puppet/modules/apache/ 目录下。

puppet module 基础模块工具目前尚未支持微软 Windows 系列操作系统。

5.3 class 类的介绍

下面来了解一下什么是 class 类。在很多编程语言中都可以看到 class（中文译为“类”）的身影，如 Java 中就有类的概念。那什么是类呢?《Java 编程思想》是这样定义的：“类是具有相同特性和行为的对象集合”。在 Puppet 中 class 类的作用是行为与资源的集合。在 Puppet 中定义 class 类通常有两种方式，一种方式是定义无参数 class 类方式，另一种方式是定义有参数 class 类方式。我们以代码片段形式来看一下这两种 class 类定义方式。

5.3.1 定义无参数 class 类

我们以修改系统的 /etc/passwd 和 /etc/shadow 文件属性为例，介绍定义无参数的 class 类方式。编辑 example.pp 文件，在文件中定义一个 example 类，其中 example 是它的类名，两个大括号中存放类的配置信息。此类完成的基本功能是通过 file 资源来修改 /etc/passwd 和 /etc/shadow 两个文件权限和所属用户。具体实现如下：

```
class example {
 file { '/etc/passwd':
  owner =>'root',
  group =>'root',
  mode =>'0644',
}
```

```
  file { '/etc/shadow':
   owner =>'root',
   group =>'root',
   mode =>'0440',
  }
}
```

这就是一个无参数的 example 类，无参数类顾名思义就是没有任何参数传入，整个配置过程由类中的 file 资源完成，类也没有任何返回状态。

5.3.2 定义有参数 class 类

我们以安装 Apache 为例介绍定义有参数的类。编辑 apache.pp 文件，在文件中通过 class 关键字定义 apache 类，并通过小括号将变量和值 $stat = 'installed' 带入 apache 类中，具体如下：

```
class apache ($stat = 'installed') {
  package {'httpd':
    ensure => $stat,
    before => File['/etc/httpd.conf'],
}
```

这是一个有参数的类，当 class 执行时就会将 $stat 变量的值 installed 带入 package 资源，来安装 Apache。

5.4 继承

Puppet 通过 inherits 关键字引入了继承的概念，继承可以让我们方便地重用代码，降低代码重复开发和改造的成本。目前 Puppet 提供两种继承方式，一种是节点继承，另一种是类的继承。两种继承使用的场景是不一样的，相信读者对 manifests 和 modules 两个目录深入了解后会深有体会。下面来分别介绍一下节点继承方式与类继承方式。

5.4.1 节点继承

节点继承方式：如 web 节点要继承一个 base 节点，base 节点包含了 web 节点所需要的基本配置信息，程序会自动由被继承的 base 节点开始执行，并依次加载 common 类和 apache 类，具体如下：

```
node base {
  include common
}
node 'web.example.com' inherits base {
  include apache
}
```

当 Agent 访问 Master 时会匹配 web.example.com，并优先加载 base 节点中的 common 类，然后加载 apache 类。这样当多个节点有共同的属性时，就可以将它们的配置抽象到一个 base 节点中，然后以每个节点分别继承这个 base 节点的方式来降低代码开发的成本。这是节点继承的优势。

5.4.2 类继承

类继承方式：base::freebsd 类继承 base::unix 类中的 file 资源。首先 base::linux 类会修改 /etc/passwd 文件信息，修改其组合用户均为 root，权限为 644，接着 base::freebsd 类会覆盖父类 file 资源的信息，将 freebad 操作系统发行版本中的 /etc/passwd 文件修改组用户为 wheel，具体如下：

```
class base::linux {
  file { '/etc/passwd':
    owner =>'root',
    group =>'root',
    mode =>'0644',
  }
}

class base::freebsd inherits base::unix {
  file['/etc/passwd'] {
    group => 'wheel',
  }
}
```

在多个类包含相同的功能时，可以将它们抽象为一个 base 基础类，并通过其他类来覆盖基础类的内容，从而达到提高代码可用率，降低开发成本的效果。这就是使用类继承的优势。

Puppet 目前不支持多重继承。

5.5 Puppet 构建主机

在上一节中我们介绍了 Puppet 构建主机的两个重要的组成部分 manifests 和 modules 目录，详细介绍了各目录的结构和存放文件的内容。本章节将延续上节内容，通过自定义的基础模块来构建主机。这里以安装配置 Apache 为例，首先介绍整体目录结构；然后介绍安装模块的 *.pp 文件；接着介绍在 site.pp 文件中加载 class 文件；最后介绍同步基础模块中的静态文件。

5.5.1 基础模块目录结构

在配置 Apache 前，首先按照基础模块目录结构来创建 httpd 的模块目录结构和 *.pp 文件。

```
# touch /etc/puppet/manifests/httpd/manifests/init.pp
# touch /etc/puppet/manifests/httpd/manifests/install.pp
# touch /etc/puppet/manifests/httpd/manifests/service.pp
# mkdir -p /etc/puppet/manifests/httpd/templates/
# mkdir -p /etc/puppet/manifests/httpd/files/
# mkdir -p /etc/puppet/manifests/httpd/tests/
# mkdir -p /etc/puppet/manifests/httpd/spec/
```

创建目录结构后，整个 Puppet 目录结构树如图 5-6 所示。

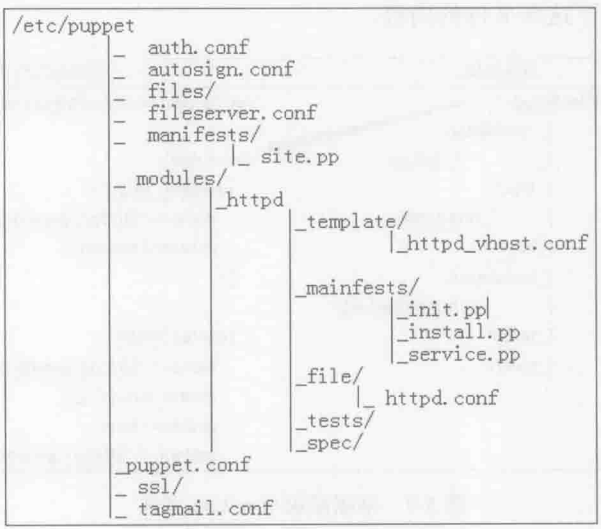


图 5-6 Puppet 目录结构树

我们简要地介绍一下基础模块（/etc/puppet/modules）目录结构。

- template 目录：httpd_vhost.conf 文件是 Apache 虚拟主机的 ERB 配置模板，在第 8 章中详细介绍。
- manifests 目录：init.pp 文件是基础模块的入口文件，入口文件名必须命名为 init.pp。它还包含了 install.pp 文件，其作用是 Apache 安装的类；service.pp 文件，其作用是 Apache 启动类。
- file 目录：httpd.conf 文件是 Apache 的守护进程配置文件，Agent 可以通过 Puppet 文件协议来下载 httpd.conf 配置文件。来看一下同步基础模块中文件的格式和案例，如表 5-1 所示。

表 5-1 Puppet 基础模块文件同步格式和案例

协 议	反斜线	模 块	模块名	下载文件
Puppet	///	Modules	my_module	service.conf
案例：puppet：///modules/my_module/service.conf				

这部分内容在第 4 章中介绍过，这里就不再详细介绍。

5.5.2 代码文件介绍

我们来依次介绍 init.pp 文件、install.pp 文件和 service.pp 文件，它们是 Apache 安装配置的代码文件，其中 init.pp 文件中定义了 httpd 类，这里读者需要注意 init.pp 文件中的类名必须与基础模块的路径名一致，如图 5-7 所示。在第 6 章中会详细介绍 Puppet 的文件导入功能。install.pp 文件中定义了 Apache 的安装类；service.pp 文件中定义了 Apache 的启动类。下面来分别看一下这些文件的内容。

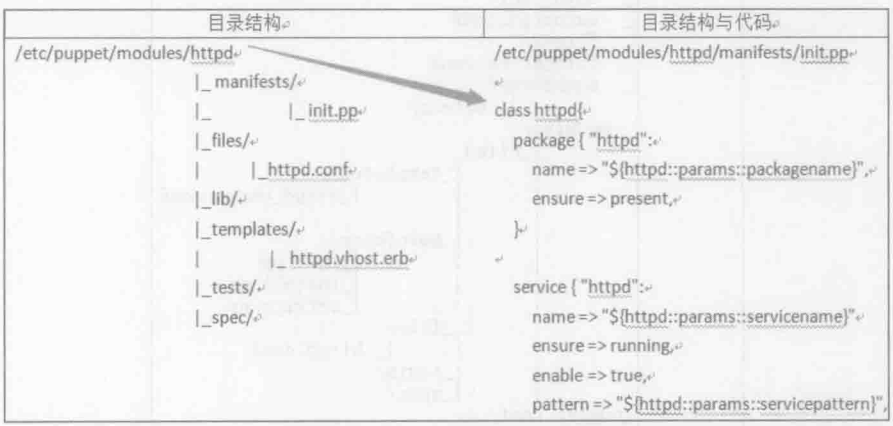


图 5-7 基础模块与 init.pp 文件

1. init.pp

/etc/puppet/modules/httpd/init.pp 文件是 Puppet 基础模块的默认加载文件，在这里可以通过 include 函数引用 install 安装类和 service 启动类，具体如下：

```
1 class httpd{
2   include httpd::install
3    include httpd::service
4 }
```

- 第 1 行：定义类名为 httpd，httpd 类名必须与基础模块（/etc/puppet/module/httpd）目录名一致。
- 第 2 ～ 3 行：通过 Puppet 内置 include 函数加载 httpd::install 和 httpd::service 类，

这里也就是将 install.pp 和 service.pp 文件中的类引入本文件中。

2. install.pp

/etc/puppet/modules/httpd/install.pp 文件是 Apache 的安装类。

```
1 class httpd::install{
2   package { 'apache2':
3     ensure => present,}
4 }
```

- 第 1 行：定义类名为 httpd::install。
- 第 2 ～ 3 行：调用了 package 资源，并设置 ensure 属性为 present，意思是安装 Apache。

3. service.pp

/etc/puppet/modules/httpd/service.pp 文件是 Apache 的启动类。

```
1 class httpd::service{
2 service {'apache2':
3   ensure => running,
4   hasstatus => true,
5   hasrestart => true,
6   enable => true,
7   require => Class["httpd::install"],}
8 }
```

- 第 1 行：定义类名为 httpd::service。
- 第 2 ～ 6 行：调用系统提供的资源 service 启动 Apache。在第 7 章详细介绍 Puppet 的资源使用方法和属性。
- 第 7 行：service 在启动 Apache 前需要先确认 httpd::install 类是否被引用，并正常执行安装好 Apache。这里的 Class 首字母需要大写，会在第 7 章中详细介绍它。

在 Puppet 中的类是单例的，它们能够在一台机器上被多个文件加载引用，但是只会被请求一次。

5.5.3 site.pp 加载配置文件

到目前为止我们已经编写好了安装配置 Apache 基础模块的 3 个重要文件 init.pp、install.pp 和 server.pp 文件，下面来看一下 Agent 如何来调用这个基础模块。编辑 /etc/puppet/site.pp 文件，将以下内容追加到 site.pp 文件中。

```
node 'example.puppet.com' {
  include httpd
}
```

当 Agent 的 Hostname 为 example.puppet.com 时访问 Master 会自动匹配 node 节点并加载 httpd 文件。这里通过 include 加载 httpd 文件，实际上 Puppet 读取的就是 /etc/puppet/modules/httpd/init.pp 文件，init.pp 文件会根据加载的 install.pp 类和 server.pp 文件，在 Agent 机器上安装 Apache，安装后启动 Apache 提供服务。

5.6 Puppet 多环境部署

伴随“云时代”和“大数据时代”的到来，运维工程师需要管理海量的服务器，每天对这些服务器需要频繁操作，推送配置、增加用户、管理 cron、清理数据和备份 / 恢复数据。通常一个运维工程师要管理上百甚至上千台服务器，一个配置推送错误会导致严重的后果，甚至会导致整个服务的不可以用，所以对于这些日常操作一定要非常谨慎。不过再谨慎也会有出错的情况发生，为了尽量避免风险，Puppet 为我们提供了多环境部署，通过多环境部署就可以实现“灰度发布”的功能。那么什么是灰度发布呢？ Puppet 又是如何实现灰度发布的呢？

在日常运维发布方式中，有一种发布叫“灰度发布”。灰度发布是指在黑与白之间，能够平滑过渡的一种发布方式。如 AB test 就是一种灰度发布方式，让一部分用户继续用 A，一部分用户开始用 B，如果用户对 B 没有什么反对意见，那么逐步扩大范围把所有用户都迁移到 B 上面来。灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度。

Puppet 提供一种功能 environments（中文译为环境），通过 puppet.conf 配置它准许将 Master 的代码配置目录进行分离，Agent 访问 Master 通过参数来区分访问不同的目录，Puppet 将这种方式管理称为环境。如将代码目录分为测试环境、开发环境、线上环境等，通过调整 Agent 的参数访问不同的 Master 环境，如图 5-8 所示。从而达到灰度发布的目的。笔者觉得通过 Puppet 的这个功能来管理海量服务器的配置让我们更加放心，也更加安全。下面来看一下如何创建环境的目录、puppet.conf 环境配置方法和 Agent 使用环境分类管理实现。

1. 创建 Puppet 环境目录

首次配置 Puppet 环境目录，需要手工创建 development（开发环境）目录、testing（测试环境）目录和 production（线上环境）目录，创建成功后接着在每个目录中再次创建 manifests 和 modules 目录。此时可以打开终端键入以下命令：

```
# mkdir -p /etc/puppet/environments/{production,testing,development}
```

```
# mkdir -p /etc/puppet/environments/production/{manifests,modules}
# mkdir -p /etc/puppet/environments/testing/{manifests,modules}
# mkdir -p /etc/puppet/environments/development/{manifests,modules}
```

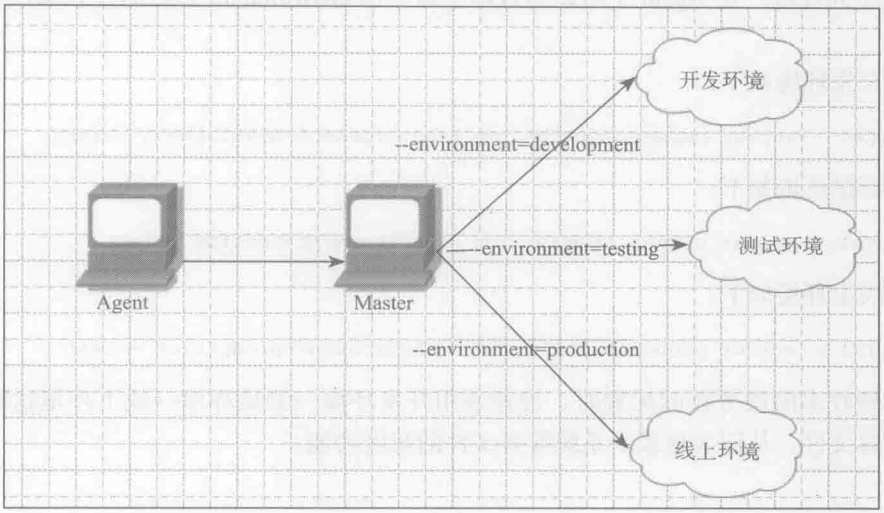


图 5-8　Puppet 的环境

2. puppet.conf 环境的配置

创建好目录后，通过向 Master 的 /etc/puppet/puppet.conf 文件追加 development、testing 和 production 的配置参数来实现分类管理。具体如下：

```
[production]
manifest = /etc/puppet/environments/production/manifests/site.pp
modulepath = /etc/puppet/environments/production/modules
[development]
manifest = /etc/puppet/environments/development/manifests/site.pp
modulepath = /etc/puppet/environments/development/modules
[testing]
manifest = /etc/puppet/environments/testing/manifests/site.pp
modulepath = /etc/puppet/environments/testing/modules
```

如 puppet.conf 配置文件内容分为 3 个环境，每个环境有自己的 manifest 和 modulepath，manifest 定义了不同环境 site.pp 文件的位置，modulepath 定义了不同环境基础模块的路径，这里也可以用“:”作为分隔来追加更多基础模块路径。

提示　追加“环境”内容到 puppet.conf 配置文件后，需要重启 Master 的守护进程，发送信号 SIGHUP 让守护进程重启，发送 SIGINT 和 SIGTERM 信号关闭 Master 的守护进程。

3. Puppet 分类管理实现

Master 配置好环境分类管理后，Agent 就可以通过参数加环境名的方式来访问统一的 Master 的不同环境。在 Agent 上可以通过以下命令加 environment 参数的方式实现访问不同的环境。

连接开发环境如下：

```
# puppet --server puppet.example.com --environment=development --test
```

连接测试环境如下：

```
# puppet --server puppet.example.com --environment=testing --test
```

连接线上环境如下：

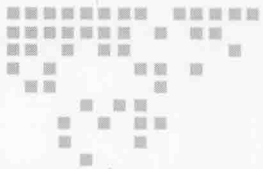
```
# puppet --server puppet.example.com --environment=production --test
```

当管理较多的服务器的配置时，推荐采用开发环境→测试环境→线上环境这样的顺序来同步配置文件，从而实现线上系统配置文件的灰度功能。

第二部分 Part 2

进 阶 篇

- 第6章 Puppet语言详解
- 第7章 Puppet 资源详解
- 第8章 Puppet ERB模板详解
- 第9章 走进Facter



Chapter 6 第 6 章

Puppet 语言详解

在第 5 章中我们曾介绍通过 Puppet 构建主机的方法，并了解了 Agent 访问 Master 的整个配置过程。同时学习了 manifests 和 modules 这两个 Puppet 的重要组成部分。在本章将继续深入介绍通过 Puppet 来构建主机，学习如何借助 Puppet 语言来构建主机。之前也曾提到，Puppet 是一款强大的配置管理工具，之所以说它强大就是因为它在延续了其他配置管理工具的基础上又融入了编程语言的概念。编程语言为后续的差异化管理配置服务器奠定了坚实的基础。同时 Puppet 提供了与高级语言近似的功能，如数据类型、作用域、变量、类、继承、命名空间和函数等，这无疑让我们通过 Puppet 来构建主机变得更加灵活方便。本章将从 Puppet 的变量讲起，首先介绍 Puppet 编程语言中的变量、变量作用域，接着介绍数据类型，然后介绍条件语句、Puppet 的函数、Puppet 的关键字、Puppet 编程规范，最后讲解 Puppet 文件的导入、命名空间与自动加载技术。

6.1 变量和变量作用域

Puppet 语言支持变量，并且变量拥有自己的作用域。在 Puppet 中的变量使用比较灵活，我们即可以自己定义变量，也可以使用 Facter 工具收集的变量，还可以使用 Puppet 自带的变量。本节将从自定义变量讲起，首先介绍在 Puppet 中如何声明变量，以及声明变量过程中的注意事项。接着介绍变量的作用域和作用域的优先级别，读者需要重视对变量优先级内容的掌握，避免后续编写 Puppet 代码由于变量优先级别导致的错误。然后再介绍 Facter 工具收集的变量，通常这些变量由 Facter 工具从 Agent 收集后传入 Master 中使用，如我们希望对来自 agent1.example.com 的 Agent 进行个性化配置，就可以通过 Facter 变量

来实现。最后介绍内置变量，这些变量由 puppet.conf 文件产生，内置变量大多包含 Master 的一些配置信息。

6.1.1 什么是变量

什么是变量？顾名思义，变量就是可变的量，在 Puppet 中，其由字母（[a ～ z] [A ～ Z]）、数字（[0 ～ 9]）和下划线（_）组成，且大小写敏感。在 Puppet 中变量必须以“$”为前缀后接“=”进行赋值，如 $test=“abc”中的 $test 就是一个变量。变量中可以保存字符串、数值、布尔值、数组、哈希和特殊的 undef 值。在大多数的编程语言中，变量在使用前要预先声明，其中 C 语言要求更加的苛刻：变量声明必须位于代码的开始部分，而在 Puppet 中无需提前声明，可以随时随地声明与使用。

下面来看一个变量使用的例子，这个例子实现的功能是通过 Puppet 提供的 file 资源将 $content 变量内容写入 /tmp/testing 文件中。编辑 site.pp 文件并追加以下内容到文件中，通过 puppet apply site.pp 来测试它的结果。

```
$content = "some content"     # 声明变量并赋值
file {'/tmp/testing':
 ensure => file,
 content => $content,
}
```

上述代码执行过程：首先 Puppet 分析文件 site.pp 中的代码是否有语法错误，然后进行编译并生成 Catalog，最后应用编译后的 Catalog。整个应用过程 Puppet 会调用 file 资源，在系统目录中创建 /tmp/testing 文件，并将 $content 变量的内容“some content”追加到 testing 文件中。可以通过 cat 命令来查看 /tmp/testing 文件的内容，具体内容如下：

```
# cat /tmp/testing
some content
```

可以看到 $content 变量的值已经追加到 /tmp/testing 文件中，这就是 Puppet 中的变量应用场景。

另外，还支持变量之间的赋值，变量之间的赋值必须使用双引号，因为双引号会解析变量中的内容后再次赋值变量。切记，如果使用单引号则无法解析变量的值。下边来介绍的 3 个典型的变量赋值的案例。

案例 1

将变量 $variable 赋值给 $value 变量，被赋值的 $variable 变量需要使用双引号引起来。

```
$variable = "1"
$value = "$variable"      # 将变量 $variable 赋值给变量 $value 必须使用双引号
```

Puppet 和很多编程语言一样，通过双引号进行解析变量中的值后再次赋值。如果使用单引号则不会解析变量值，直接将变量名进行赋值。以下为两种赋值方式的结果：

```
$value = "$variable"  # 结果为 1
$value = '$variable'  # 结果为 $variable
```

案例 2

在包含字符串的变量赋值变量时，推荐使用大括号“{}”，以便更好地识别变量名。

```
$one = "1"
$value = "this is number:${one}"     # 在包含字符串的 $value 变量中使用 $one 变量通常书写方式为 ${one}
```

案例 3

需要大家注意的是，大部分编程语言中变量是可以重复赋值的，即变量 $content 可先赋值为“some contents”，然后再次赋值为“content”。但在 Puppet 中的变量不能重复赋值。来看一个代码片段。

```
$content = "some content"       # 首次赋值 $content 变量
file {'/tmp/testing':
 ensure => file,
 content => $content,
}
$content = "content"     # 再次赋值 $contents 变量
```

以上代码片段重复赋值的执行结果会导致以下 Puppet 的一个错误。为什么会这样呢？

```
Err:Cannot reassign variable location at /etc/puppet/manifests/site.pp:2
```

因为 Puppet 具有解释性语言的特性和动态的作用域，所以它在一个作用域内不准许变量重复赋值。那什么是作用域呢？让我们继续看下一小节。

6.1.2 变量作用域

在 Puppet 代码执行过程中，变量并非在所有的代码范围中都是有效可用的，而限定这个变量使用有效范围的就是作用域。在 Puppet 中每创建一个 class 类、定义 node 节点都会引入一个新的作用域。目前 Puppet 的作用域被划分为 3 类：top 作用域、node 节点作用域和 local 作用域。下面我们通过代码片段来了解一下这 3 类作用域的使用方法。

1. 作用域的作用范围

Puppet 中 3 类作用域作用范围的示意如图 6-1 所示。

- 同一个作用域中不可以声明相同的变量名。即 top 作用域、node 节点作用域和 local 作用域声明的变量名不能重复。
- top 作用域中的变量只在 top 作用域中有效。
- node 节点作用域中的变量只在 node 节点中有效。但 node 节点作用域还可以访问 top 作用域的变量。

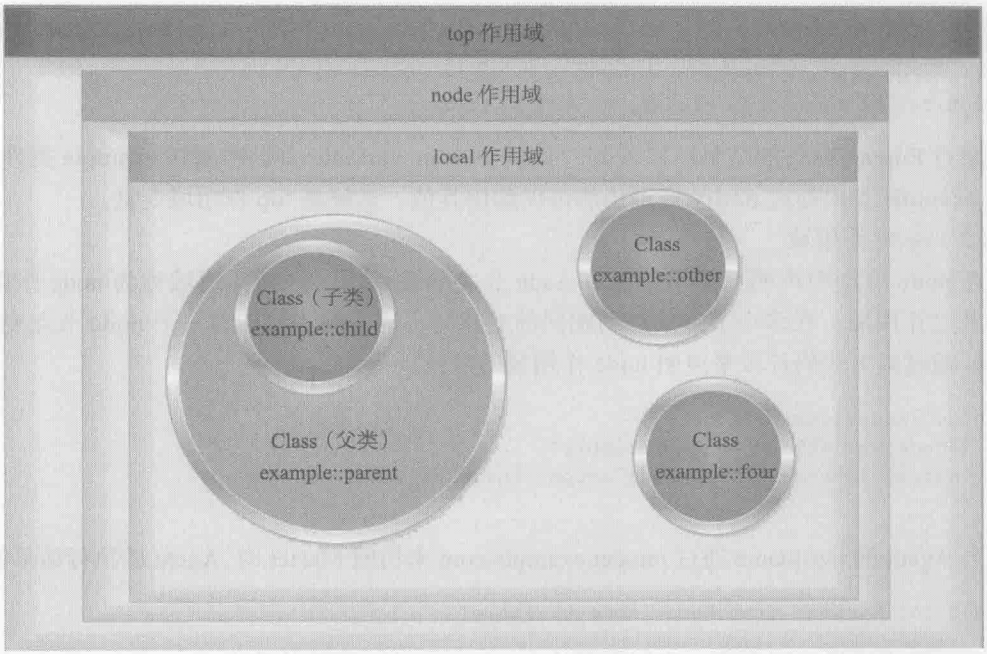


图 6-1 Puppet 作用域

- local 作用域变量只在 local 作用域中有效。但 local 作用域可以访问 node 节点作用域和 top 作用域中的变量。
- example::parent 类、example::other 类和 example::four 类可以使用在本身 local 作用域中的变量、node 作用域的变量和 top 作用域的变量，但是它们之间的变量是不可以互相访问的。
- example::child（子类）可以访问 example::parent（父类）作用域、node 节点作用域和 top 作用域变量。但 exmple::child（子类）不能访问 example::other 类和 example::four 类作用域的变量。

下面就来通过示例介绍 Puppet 中 3 类作用域的作用范围及其变量的定义方法。

（1）top 作用域

声明变量后可以在 class 类内和 node 节点内调用的作用域被称为 top 作用域（也可以理解它为全局作用域，因为它在任何 node 节点和 class 类中都可以使用）。通过以下代码片段来声明 top 作用域的变量。

```
# site.pp
$top_variable = "top_variable"
class example {
  notify {"Message from elsewhere: $top_variable":}
}
```

通过 puppet apply site.pp，执行结果如下：

```
# puppet apply site.pp
notice: Message from elsewhere: top_variable!
```

通过 Puppet 输出的结果可以看到，代码中 $top_variable 变量声明在 example 类外，但是在 example 类内通过 notify 资源仍然可以调用其值，这就是 top 作用域变量。

（2）node 作用域

在 node 节点中声明变量后只能在 node 节点中被调用，这种作用域称为 node 作用域，又称节点作用域。在多个节点中声明相同的变量时，同一时间只能有一个 node 节点变量被调用。通过以下代码片段来声明 node 作用域的变量。

```
node 'puppet.example.com' {
  $node_variable = "node_variable"
  notify {"Message from node scope: $node_variable":}
}
```

当 Agent 的 Hostname 通过 puppet.example.com 来访问 Master 时，Agent 的执行结果如下：

```
notice: Message from here: node_variable!
```

通过输出结果可以看到，我们声明的 $node_variable 变量的值为“node_variable!”，表示声明了 node 作用域的变量。

（3）local 作用域

变量声明后只能在 class 类内使用，这种作用域称为 local 作用域或本地作用域。通过以下代码片段来声明 local 作用域的变量。

```
# site.pp
class scope_example {
  $local_variable = "local_variable"
  notify {"Message from here: $local_variable":}
}
```

执行 puppet apply site.pp，执行结果如下：

```
# puppet apply site.pp
notice: Message from here: local_variable!
```

通过输出结果可以看到，我们声明的 $node_variable 变量的值为“local_variable!”，表示它为 local 作用域的变量。

2. 作用域的优先级

变量的作用域与作用域之间是有优先级的，当相同的变量名出现在同一代码文件中时，它的优先级应该是 local 作用域最高，node 作用域次之，top 作用域级别最低。通过以下代码片段来看作用域与作用域之间的优先级顺序（其中“#”表示文件注释）。

```
# site.pp
$variable = "top scope"      # top 作用域
node 'puppet.example.com' {
  $variable = "node scope"      # node 作用域
     class scope_example {
         $variable = "local scope"     # local 作用域
          notify {"Message from here: $variable":}
     }
}
```

当 $variable 变量在 local 作用域、node 作用域和 top 作用域同时出现时，$local 作用域会覆盖以上两个 $variable 的变量。执行 puppet apply site.pp，执行结果如下：

```
notice: Message from here: local scope
```

执行代码片段后输出结果为“local scope”，证明了当 3 种作用域同时出现时，local 作用域的优先级别最高。如果这个例子中没有 local 作用域的变量，只有 node 作用域与 top 作用域的变量，那么 node 作用域的变量就会覆盖 top 作用域变量中的值。此例子也同时解答了 6.1.1 节最后的问题，声明变量时如果不更改变量名，需要通过作用域将它们分开，才能避免 Puppet 的报错。

6.1.3 Facter 变量

Facter 是一款扩展性强且功能强大的跨平台的系统性能分析收集工具，它可以收集 Agent 的信息，并将收集到的 Agent 信息作为变量传给 Master 使用。在 Puppet 中这些变量均为 top 作用域变量，可以在 Puppet 语言中的任何位置调用到它们。通过系统终端输入 facter 命令看到这些变量。由于后续章节中会用到 Facter 变量，所以这里介绍一些 Facter 的常用变量，如表 6-1 所示。Facter 工具有很强的扩展功能，并支持通过 ymal 格式来导入更多变量。本章只介绍了它的冰山的一角，关于 Facter 工具更多的信息会在第 9 章中详细介绍。

表 6-1 Puppet 常用系统变量

变 量 名	变量含义
ipaddress	IP 地址
kernel	内核版本
memorysize	内存
operatingsystem	操作系统发行版本
rubyversion	Ruby 版本
uptime	开机时间
hostname	系统 Hostname

以下为调用 Facter 变量的代码片段（这里以调用本机 IP 地址为例）：

```
# notify.pp
notify{"$ipaddress":}
```

执行 puppet apply notify.pp，执行结果如下：

```
notice: 192.168.7.77    # 其中 "192.168.7.77" 为 $ipaddress 的值
notice: /Stage[main]//Notify[192.168.7.77]/message: defined 'message' as
'192.168.7.77'
notice: Finished catalog run in 0.02 seconds
```

可以看到 $ipaddress 变量中存放的为本机的 IP，它由 Facter 收集并传给 Puppet 中作为变量使用。

6.1.4 内置变量

Puppet 还为我们提供了一些内置变量，由 puppet.conf 配置文件定义，并在 Puppet 代码文件中可以直接使用的变量称为内置变量。通常内置变量经常与 Puppet 语言的分支语句结合使用。这些内置变量分为 Agent 端的内置变量和 Master 端的内置变量两种。

Agent 端内置变量如下。

- $enviromnent：在第 5 章中我们曾介绍了 Puppet 的环境。通过 $enviroment 可以查到相关的环境信息。
- $clientcert：Agent 的 certname 信息。

Master 端内置变量如下。

- $servername：Master 的 fqdn。
- $serverip：Master 的 IP。
- $module_name：Master 中的模块名。

6.2 数据类型

数据可以按照数据结构的大小进行分类，并将分类后的结果存放到内存中，我们将数据进行分类后的结果称为数据类型，数字和字符串都是常用的类型之一。目前 Puppet 数据类型支持字符串类型、数值类型、数组、哈希、布尔值、undef 和正则表达式，如图 6-2 所示。下面以代码片段的方式来介绍一下 Puppet 的数据类型和注意事项。

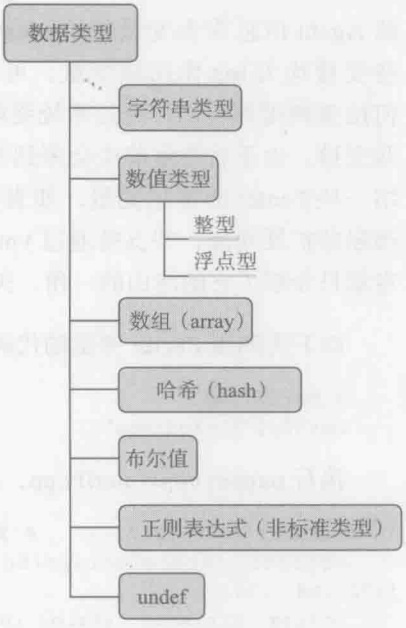


图 6-2 Puppet 数据类型

6.2.1 字符串类型

定义字符串类型需要以双引号（“”）或单引号（‘’）进行声明，在 Puppet 中默认的数据类型就是字符串类型。字符串类型声明时需要注意以下 3 点。

- 不能使用 Puppet 关键字（见 6.5 节）。
- 字符串类型由字母（[a ～ z][A ～ Z]）、数字（[0 ～ 9]）、连接符（-）和下划线（_）组成。
- 官方网站建议字符串类型使用 UTF-8 的字符集，这也是为后续与 Puppetdb 结合做准备。

以下两行通过都是通过双引号或单引号声明有效的 Puppet 字符串类型。

```
$variable1 = 'hello world'
$variable2 = "hello world"
```

在 Puppet 中通过单引号声明的字符串中，如果遇见尾部有“\”的情况，则需要通过“\”进行转义。

```
path => 'C:\Program Files(x86)\\'
```

当我们使用一些特殊的符号作为变量值时，需要将它们转义后再使用。字符串转义序列如表 6-2 所示。

表 6-2 转义序列

转义序列	含 义
\$	$ 符号
\"	双引号
\'	单引号
\\	反斜杠
\n	回车换行符
\r	回车换行
\t	tab 键，一个 tab 键默认是 7 个空格
\s	空格

6.2.2 数值类型

在 Puppet 语言中支持数值类型，数值类型是指定义成的数值形式的数据，这种数据可以直接进行加、减、乘、除等数学运算。在 Puppet 中数值类型包括整数类型与浮点类型，与此相对应，Puppet 中的数值类型支持整数类型的运算也支持浮点类型的运算。下面就来详细介绍一下。

整数类型的计算方式：

```
$product = 8 * 4
$product = 7 + 5
```

浮点型的计算方式：

```
$product = 8 * 0.12
```

读者需要注意数值类型的书写格式。以下是错误的语法格式：

```
$product = 8 * +4     # 错误的语法格式
$product = 8 * .12    # 错误的语法格式
```

以上错误中包含：

- 符号不要同时使用。
- 浮点型数据以数字加小数点形式表述，正确的方式如 0.12。

目前 Puppet 数值类型的运算符包括加、减、乘、除、与、或、取反等。不同的运算符之间是有优先级的。把 Puppet 中的运算符按照优先级按从高到低的顺序进行排列，得到如表 6-3 所示的结果。

表 6-3 Puppet 运算符优先级

运 算 符	说 明
!	取反
In	范围
* 和 /	乘和除
- 和 +	减和加
<< 和 >>	左移和右移
== 和 !=	等于和不同于
>= <= 和 > <	大于等于、小于等于和大于、小于
And	与
Or	或

6.2.3 数组

在 Puppet 语言中的支持数组类型。所谓数组就是将同一类事物按照一定的顺序放到一个集合中，通过定义这个集合来完成对数组中所有事物的定义。Puppet 中的数组通过方括号来定义，数组中的数据通过“,”分隔。灵活应用数组类型进行变量的定义，可以缩短和简化程序，提高代码的可读性。下面就来介绍数组的定义、取值和追加数组。

1. 数组定义

Puppet 通过 [" 数组内容 1", " 数组内容 2"] 的形式来定义数组，并将定义后的数组赋值给变量。下面以安装 Ruby 的相关环境为例，介绍定义 Puppet 数组。首先将 Ruby 环境的软件包以“,”形式进行分隔，然后将声明的数组赋值给 $packages 变量，最后调用系统 package 资源对 $package 数组变量中定义序列进行依次加载并安装。以下为代码片段：

```
$packages = [ "ruby1.8-dev",     # $package 数组中声明了要安装的软件包
"ruby1.8",
"ri1.8",
"rdoc1.8",
"irb1.8",
"libreadline-ruby1.8",
"libruby1.8",
"libopenssl-ruby" ]
package { "$packages": ensure => installed }    # 通过 package 资源来依次安装 $package 数组中内容
```

Puppet 通过系统 package 资源对 $package 数组的内容依次进行下载和安装，这也是数组常见的应用场景。

2. 数组的取值

再来看一下数组如何取值。可以将数组赋值给变量，并通过 [数组下标] 的方式进行取值。Puppet 数组和其他编程语言中的数组一样，下标从 0 开始。下面来看一个例子。

首先定义数组，并将定义好的数组值赋值给 $foo 变量，调用 notice 函数打印数组中的内容。编辑 array.pp，具体内容如下：

```
$foo = [ 'one', 'two', 'three' ]
notice( $foo[1] )  # 数据下标以 0 开始，通过 notice 函数打印 $foo 数组 two 值，对应值为下标 1
```

通过 puppet apply apply.pp 来查看结果。

```
puppet apply array.pp
notice: Scope(Class[main]): two     # 打印 two 数组值
notice: Finished catalog run in 0.01 seconds
```

notice 会将结果输出到屏幕，它会取 $foo 数组中的第二个值，输出结果为 two。

3. 数组追加

当我们定义数组后，仍然可以通过“+=”符号向数组中追加新值。如以下例子，在不同的类中需要增加本身类的属性时可以通过“+=”符号来追加。

```
$package = ['base' ,'php']
class nginx {
   $packages += ['nginx']     # 追加后，$packages 数组中包含了 ['base', 'php','nginx']
}
class apache {
   $packages += ['apache']    # 追加后，$packages 数组中包含了 ['base', 'php','apache']
}
```

需要注意的是通过“+=”符号在追加过程中数据类型要一致，如将字符串型赋值给数组，这是错误的使用方法。“+=”除了可以追加数组外，还可以在字符串和哈希类型中使用。

4. 嵌套数组

Puppet 通过嵌套数组来模拟多维数组，并通过索引的方式访问嵌套数组的值。编辑 array.pp，具体内容如下：

```
$foo = [ 'one', {'second' => 'two', 'third' => 'three'} ]
notice( $foo[1]['third'] )
```

以上代码为打印 $foo 嵌套数组的 third 键对应的值。通过 puppet apply apply.pp 来查看结果。

```
# puppet apply array.pp
notice: Scope(Class[main]): three    # 打印 three 数组值
notice: Finished catalog run in 0.01 seconds
```

6.2.4 哈希类型

Puppet 语言支持 hash（中文译为哈希，下称哈希值）类型。哈希值是近来非常流行的数据类型，它与数组类似，都是带索引的对象集合，与数组的区别在于作为索引不仅限于数字，可以是任何对象。目前比较有代表性的软件有 Memcached、Berkeley DB 和 Redis，它们均采用了哈希值的数据类型存储方式。同样 Puppet 中的哈希也使用“键 / 值”为格式组成的键值对。在 Puppet 中哈希类型的“键”必须是字符串类型，但其“值”可以是任意的类型。下面来看一下哈希的格式和取值方法。

定义哈希的类型键为 key1，值为 val1 ，如果设置多个“键 / 值”需要用逗号分隔。实现的代码如下：

```
{ key1 => 'val1', key2 => 'val2',…}
```

定义变量并将定义好的哈希列表赋值给 $myhash 变量，调用系统 notice 函数输出

$myhash 变量中键对应的值。实现代码如下：

```
$myhash = { key => "some value",
            other_key => "some other value" }
 notice( $myhash[key] )
```

以上输出结果为 key 对应的值 some value。

6.2.5 布尔类型

Puppet 语言支持 Booleans（中文翻译“布尔”类型，下称布尔型），布尔型经常用于函数的返回状态，布尔型只有两个值 true（真）或 false（假）。在通过布尔型赋值变量时，需要注意不要加双引号（“”）或单引号（‘’）符号，示例如下。

```
$switch = true     # 正确的赋值方法
$switch = false    # 正确的赋值方法
$switch = 'true'   # 错误的赋值方法
$switch = "false"  # 错误的赋值方法
```

如以下代码片段中的 str2bool 函数的返回值就是布尔型，通过 str2bool 函数判断期返回值，如果是 true 则加载 ntp::disabled 类文件。

```
if str2bool($is_virtual) {     # 判断如果 $is_virtaul 返回值为 true 则加载 ntp::disabled 类
  include ntp::disabled
}
```

需要注意，str2bool 函数需要安装 puppetlabs-stdlib 模块。在 Puppet 的 puppetlabs-stdlib 模块中提供了两个函数 str2bool 和 num2boole。其中 str2bool 可以将字符串转换为布尔型，num2boole 可以将数值型转为布尔型，两个函数的功能类似于很多编程语言中的强制类型转换。

在使用 str2bool 和 num2boole 这两个函数时需要提前 puppet module install puppetlabs-stdlib 安装这个模块所在的库文件才可以使用。

6.2.6 正则表达式

Puppet 语言支持 Regular Expression（中文译为“正则表达式”，下称正则表达式）。在处理字符串时经常有查找或替换某些字符串的需求，正则表达式就是用于描述这些规矩的工具。在 Puppet 中支持标准 Ruby 正则表达式。但是读者需要注意正则表达式并非 Puppet 标准的数据类型，它不能被赋值，也不能用于函数之间的传递。来看一下常用的正则表达式标记号和使用案例。

正则表达式常见的标记号如下。

- []：用于描述范围（如 [a-z]，表示范围 a~z 之间）。
- ()：用于包含正则表达式。
- \w：用于描述字母或数字，相当于 [0-9a-zA-Z]。
- \W：非字母或数字。
- \s：匹配 [\t\n\r\f]，其中（\t）为制表符、（\r）为回车符、（\n）为换行符、（\f）为换页符，\s 表示匹配这些符号的简写方式。
- \S：匹配非空字符。
- \d：匹配 [0-9] 数字。
- \D：匹配非数字。
- \b：匹配退格符。
- \B：非字边界。
- *：前面元素出现 0 次或多次。
- +：前面元素出现 1 次或多次。
- {m，n}：前面元素最少出现 m 次，最多出现 n 次。
- ?：前面元素最多出现 1 次，等价于 {0,1}。
- |：与前面或后面表达式匹配。

以上为 Puppet 的标记号，通过以下代码片段来看一下正则表达式的使用方法。

```
$packages = $operatingsystem ? {
   /(?i-mx:ubuntu|debian)/          => 'apache2',
   /(?i-mx:centos|fedora|redhat)/ => 'httpd',
}
```

代码片段表示判断 $operatingsystem 操作系统发行版本，如果是 ubuntu 或 debian 系统发行版本则安装 apache2 包，如果是 centos、fedora 或 redhat 系统发行版本则安装 httpd 包。其中 i、m、- 和 x 为 Puppet 特有。具体内容如下：

- i 表示忽略大小写。
- m 表示把“.”当做换行符使用。
- - 表示不使用某转移符号。
- X 表示忽略模式中的空白字符和注释。

下面来看两个案例，第一个案例是根据 UNIX/Linux 系统匹配不同系统的发行版本，并根据发行版本设置 /etc/passwd 文件组权限；第二个案例是通过正则表达式来进行 IP 替换，将本机 IP 替换为 C 段的 IP。

案例 1

通过识别不同的 UNIX/Linux 系统发行版本，将 /etc/passwd 文件设置为不用的组用户。这里用 selector 表达式来判断 $operatingsystem 系统变量的值是什么发行版本，如果是 darwin 或 FreeBSD 系统就将 users 赋值给 $rootgroup 变量。这里用到了正则表达式的

(Darwin|FreeBSD) 功能，假设系统发行版本是 FreeBSD 就会被它匹配到，最终 file 资源会将 /etc/passwd 文件组权限设置为 users 值。以下为代码片段：

```
$rootgroup = $operatingsystem ? {
  /(Darwin|FreeBSD)/ => 'users',  # 通过正则表达式方式判断操作系统发行版本是 Darwin 或 FreeBSD
          default => 'root',
}
file { '/etc/passwd':
  ensure => file,
  owner  => 'root',
  group  => $rootgroup,
}
```

案例 2

我们希望通过 Puppet 语言将本机的 IP 替换成一个 C 段的 IP，并将替换后的 C 段 IP 与本机 IP 一并输出。这里就需要通过正则表达式来替换 IP 末位，将末位转为 0。编辑 conv_ip.pp，将以下代码追加到文件中。

```
$class_c = regsubst($ipaddress, "(.*)\..*", "\1.0")
notify { $ipaddress: }
notify { $class_c: }
```

其中 $ipaddress 变量由 Facter 收集，可以在 Puppet 代码中直接使用，$ipaddress 的值为本机系统的 IP。regsubst 为系统的替换函数，它需要以“,”作为分隔输入 3 个参数值，分别是源字符串、匹配模式（pattern）和替换结果。通过 regsubst 函数将替换后的结果传给 $class_c 变量，并通过 notify 资源打印出源 IP 和替换后的 IP。通过 puppet apply conv_ip.pp 看一下结果。

```
notice: 10.168.7.14      # $ipaddress 值
notice: /Stage[main]//Notify[10.168.7.14]/message: defined 'message' as
'10.168.7.14'
notice: 10.168.7.0       # 经过 regsubst 函数替换后的 $class_c 值
notice: /Stage[main]//Notify[10.168.7.0]/message: defined 'message' as
'10.168.7.0'
notice: Finished catalog run in 0.04 seconds
```

可以看到本机的源 IP 为 10.168.7.14，替换后的 IP 为 10.168.7.0。我们指定了“(.*)\..*”作为匹配模式，“\1.0”作为替换结果，匹配模式将匹配整个 IP 地址，捕获的前 3 个字节放在一对圆括号内，被捕获的文本可以在替换结果中使用“\1”来引用。被匹配的全部文本 IP 为 10.168.7.14 将使用替换结果（replacement）来替换，这里是“\1”（从源字符串中捕获的文本）跟上字符串“.0”，最终获得 C 段 IP:10.168.7.0。notify 资源将替换前后结果输出到屏幕上。

6.2.7 undef

undef 值在 Puppet 是一个特殊的值，它等同于 Ruby 语言中的 nil。没有声明过但是仍

然可以使用的值就属于 undef 值。以下为 undef 的通常用法：

```
 $apache = $operatingsystem ? {
    centos             =>'httpd',
    redhat             =>'httpd',
/(?i)(ubuntu|debian)/ =>'apache2',
     default           => undef,    # 如果没有匹配到任何操作系统发行版本，则默认设置为 undef 值
}
```

通过 $operatingsystem 来判断系统的发行版本，如果匹配不到 selector 语句中的操作系统发行版本，就可以将 undef 值赋值给 $apache 变量。

6.3 条件判断语句

条件判断语句是编写复杂程序的基础，目前 Puppet 2.7 以上版本支持 if…elsif…else、selecter 和 case 等条件语句。它们的条件判断分为两类：一类为条件执行（if…elsif…else、selecter）通过逻辑判断来选择要执行的特定代码或加载代码；另一类（case）是循环执行类，依照代码规则来执行特定代码或加载代码。下面来分别介绍一下这两类条件语句的基本语法与案例。

6.3.1 if…elsif…else 条件语句

首先来了解一下 if…elsif…else 条件语句的基本语法。

```
if condition1 {
  block1 of code
}
elsif condition2 {
  block2 of code
}
else {
  block3 of code
}
```

以上为 if…elsif…else 条件判断语句的语法和大致判断流程。它首先判断条件表达式 condition1 是否为 true，如果为 true 则执行 block1 语句；如果不为 true 则继续判断第二个条件表达式 condition2 是否为 true，如果为 true 则执行 block2 语句；如果仍然不为 true 则执行 block3 语句。在 Puppet 中 if…elsif…else 条件判断语句较为常见，如判断操作系统的发行版本，并根据不同的版本加载不同的配置文件；或者通过 if…elsif…else 判断条件的范围等。下面来介绍 3 种 if…elsif…else 条件判断语句。和很多的编程语言不同之处，Puppet 除了支持常见的判断外，if…elsif…else 还支持通过正则表达式或 in 取范围方式来判断返回值。

案例 1

以下是条件判断语句 if…elsif…else 的使用案例。判断系统发行版本，根据系统发行版

本加载不同的基础模块文件。

```
if $operatingsystem == 'Ubuntu'
  { include httpd }
 elsif $operatingsystem == ' RedHat'
  { include memcache }
else
  { include ntp }
```

首先判断系统 $operatingsystem 变量是否等于 Ubuntu，如果是则加载 httpd 类文件；否则判断系统 $operatingsystem 变量是否等于 RedHat，如果是则加载 memcache 类文件；如果还不匹配的话最后调用加载 ntp 类文件，整个判断结束。以操作系统发行版本为 Ubuntu 为例，条件判断语句会加载 httpd 类，并结束整个条件的判断。

案例 2

通过 if…elsif…else 与正则表达式的结合来判断条件的范围。以 $hostname 变量为例，它是 Agent 机器的 Hostname，判断它的范围是否以 www 开始，并且后接数字。如果判断为真则显示欢迎信息。代码片段如下：

```
if $hostname =~ /^www\d+/ {
  notice("Welcome to web server number: $hostname")
}
```

以 Agent 的 Hostname 为 www27.exaple.com 为例，最终输出结果为 Welcome to web server number：www27.exaple.com。

案例 3

以下通过 in 取范围方式选择最终结果。

```
if $operatingsystem in ["SUSE","Redhat"] {
  notice("the system is :$hostname")
}
```

通过 in 在数组范围中匹配最终结果，如果匹配到则打印提示语为 the system is SUSE。

6.3.2 case 语句

Puppet 语言中的 case 语句和 if…elsif…else 完成的结果一样，但是在多重条件判断语句时使用 case 语句代码要比 if…elsif…else 代码更加整洁些。下面来看一下 case 语句的使用方法。

通过 case 语句来判断操作系统的发行版本，并加载不同的类文件。代码片段如下：

```
case $operatingsystem {
  'Solaris':              { include role::solaris }
  'RedHat', 'CentOS':  { include role::redhat  }
```

```
    /^(Debian|Ubuntu)$/: { include role::debian }
    default:             { include role::generic }
  }
```

首先判断系统 $operatingsystem 变量的值，如果值是 Solaris 则加载 role::solaris 类；如果值是 RedHat 或 CentOS 则加载 role::solaris 类；如果值是 Debian 或 Ubuntu 则加载 role::debian 类；如果都没有匹配到，则最终加载 role::generic 类。假设我们的操作系统是 RedHat，case 语句会匹配到此操作系统发行版本，并加载 role::redhat 类文件，整个 case 语句流程结束。

6.3.3 selector 语句

在很多编程语言中我们经常可以看到 selector 表达式的身影。它和 C 语言或 Ruby 语言中的三元操作类似，意思是在两个选项中任选其中一个赋值。下面是 Puppet 语言中的 selector 表达式的使用方法，代码片段如下：

```
$rootgroup = $operatingsystem ? {
                'Solaris' => 'wheel',
      /(Darwin|FreeBSD)/ => 'users',
                default => 'root',
    }
file { '/etc/passwd':
  ensure => file,
  owner  => 'root',
  group  => $rootgroup,
}
```

首先根据 $operatingsystem 变量判断系统的发行版本，并根据发行版本自动匹配 /etc/passwd 不同的组用户权限。接着声明 $rootgroup 变量，通过 selector 表达式判断系统 $operatingsystem 变量中的操作系统发行版本是否与哈希表中某 key 匹配。如果匹配则将相应操作系统版本为 key 中的值传给 $rootgroup 变量，最后通过系统 file 资源重新设置 /etc/passwd 的文件组权限为 $rootgroupt 变量值。

假设我们的操作系统发行版本是 SUSE，哈希表中并没有 SUSE 发行版本，selector 会默认匹配 root 值，并将值赋值给 $rootgroup 变量。最后 file 资源根据 $rootgroup 变量将 /etc/passwd 文件的 owner 与 group 设置为 root，整个 selector 语句结束。

6.4 Puppet 函数介绍

函数的主要用途是完成一个功能的集合。截至本书出版前，Puppet 共提供了 43 个函数，通过这些函数为系统管理员配置管理系统节约了很多时间和精力。但想通过这 43 个函数来解决运维中面临的所有的问题是不可能的，即 98 法则中所说，它只能解决 98% 用户的需

求，但还有 2% 的用户需求是无法满足的，而这 2% 用户很有可能是一些高端的用户。这时我们就可以通过以下两种方式来解决。

方式一：通过官方网站获取已经扩展好的工具包，工具包中包含扩展好的自定义函数、资源、提供者和 Facter 扩展。笔者推荐这种方式，因为这种方式最简便，而且实现成本也最低。

方式二：通过 Ruby 语言开发扩展程序。但这种适合比较的特殊的情况，如我们需要的功能本身 Puppet 和外部扩展均不提供，所以只能由自己独立来开发。但是在开发之前建议优先了解现有的程序逻辑，熟悉它的代码结构、使用方式和工作原理，为我们后续独立开发开拓思路奠定基础。

本节主要介绍 Puppet 自带的系统函数，关于官方网站扩展包中的函数和通过 Ruby 扩展函数相关内容会在第 10 章中详细介绍。关于 Puppet 函数的更多信息请参考官方网站 http://docs.puppetlabs.com/references/latest/function.html。

6.4.1 常用系统函数

截至本书出版前，Puppet 共提供了 43 个系统函数，它分为 6 类，如表 6-4 所示。由于函数比较多，不能逐一详细介绍，这里只介绍常用的系统函数的使用方式，其他的按类划分简单介绍，在后续章节用到相关函数时，再详细介绍它们的使用方法。

表 6-4 Puppet 系统函数

常用	define、tag、tagged、generate、template、realize、regsubst、require、include、versioncmp
日志级别	alert、crit、emerg、notice、warning、debug、err、info
数字签名	md5、sha1
hiera	hiera_array、hiera_hash、hiera_include
其他	each、split、sprintf、contain、create_resources、extlookup、fail、file、filter、fqdn_rand、inline_template、lookup、map、reduce、search、shellquote、slice
已废弃	select、collect

1. define

define 函数主要用于创建自定义函数，define 支持参数但不支继承。通常可以通过 define 函数将多个资源整合为一个资源。下面来了解一下 define 函数的格式和案例。

（1）define 函数格式

以下为 define 函数的格式：

```
define 函数名 ($variable1,$variable2) {
}
```

define 函数后接自定义函数的名与大括号，大括号中可以包含变量、资源、判断语句和 class 类等。

（2）define 函数案例

通过 define 函数来创建 create_user 自定义函数并整合创建系统组和账户的功能。首先通过 define 创建 create_user 自定义函数，在 create_user 自定义函数参数中声明 $name 和 $path 变量，然后在 create_user 自定义函数内调用 group 和 user 资源来创建组和账号（关于资源会在第 7 章详细讲解，这里只需要了解这两个资源的作用就可以）。编辑 create_user.pp 文件并将以下内容追加到文件中。

```
define create_user ($user = $name,$path) {
   # group 资源与 user 资源顺序不能颠倒
   group {"$name":      # 创建系统 " 组 " 资源且 $name 需要用双引号
     ensure => present,
   }
   user {"$name":       # 创建系统 " 账号 " 资源且 $name 需要用双引号
     ensure => present,
     home => $path,
   }
}
```

最后再来看一下如何使用已经创建好的 create_user 自定义函数。将参数以 hash 的“键 / 值”形式传入函数，其中“键”为函数中声明的参数，“值”为参数值。将代码追加到 create_users.pp 文件行尾，并调用 puppet apply create_user.pp 文件。代码如下所示：

```
create_user { 'create_user':
    name => 'tony',
    path => '/home/tony',
}
```

执行结果：create_user 自定义函数先创建 tony 组，接着创建 tony 账户和账户的宿主目录 /home/tony，这样通过 create_user 自定义函数就将创建系统账户和组融合到了一起。

2. tagged 函数介绍

有时 Puppet 的一个类需要知道另一个类是做什么的，例如一个数据库类需要知道一个节点是否是 Web 服务器节点，就可以通过 tag 公有资源做标记，并通过系统 tagged 函数判断被标记的类与类之间的关系。下面让来通过代码片段了解一下 tagged 函数与 tag 公有资源的用法。

首先创建两个节点，在两个节点中通过系统 tag 函数设置标识，分别为 web 和 databaseserver，然后在 basics 类中通过 tagged 函数判断类加载的 tag 名是否存在。

```
class basics {
   if tagged(webserver) {     # 通过 tagged 函数判断被加载类的标识是否为 webserver
    notice("This is a web server")
   }
   if tagged(databaseserver) {
   notice("This is a databaseserver")   # 通过 tagged 函数判断被加载类的标识是否为 databaserver
```

```
    }
}
node 'node1.testing.com' {
  tag(webserver)      # 通过 tag 标识此节点为 webserver
  include webserver
}
node 'node2.testing.com' {
   tag(databaseserver)      # 通过 tag 标识此节点为 databaserver
   include databaseserver
   include basics     # 加载 basics 基础类
}
```

3. generate 函数

generate 函数可以调用 Master 端的系统命令，并将输出结果回传给 Agent。以下为代码片段。

```
$result = generate('/usr/bin/env', 'bash', '-c','ifconfig')
notify $result
```

generate 中的参数以逗号作为分隔，其参数含义如下：

- /usr/bin/env 表示加载环境。
- bash 表示用的 shell 解析器。
- -c 为 bash 的参数，表示后接字符串。
- ifconfig 为系统命令，表示查看机器的 IP 地址。

4. template

template 函数可以通过 file 资源调用 ERB 模板（ERB 模板在第 8 章中详细介绍）。以下为调用代码片段。

```
template('my_module/template.erb')
```

它还可以合并模板。以下为代码片段。

```
template('my_module/template1.erb','my_module/template2.erb')
```

5. versioncmp

versioncmp 函数用于版本号间的比较。以下为代码片段。

```
$result = versioncmp(a, b)
```

versioncmp 函数有 3 个返回值，值和含义如下：

- 如果版本 a 大于版本 b，返回值就为 1。
- 如果两个版本相等，返回值就为 0。

❑ 如果版本 a 小于版本 b，返回值就为 -1。

下面看一个 versioncmp 案例：

```
if versioncmp('2.6-1', '2.4.5') > 0 {
    notice('2.6-1 is > than 2.4.5')
}
```

versioncmp 函数判断“2.6-1”与“2.4.5”的版本号的大小，这里前者大于后者，所以输出“2.6-1 is > than 2.4.5”信息。

6. reduce 函数

reduce 函数接收 Puppet 的数组与 hash 值，并可以累加输出。以下为代码片段。

```
# 数组
$a = [1,2,3]
$a.reduce |$memo, $entry| { $memo + $entry }    # 将 $a 数组中的值进行累加输出
#=> 6
# hash
$a = {a => 1, b => 2, c => 3}
$a.reduce |$memo, $entry| { [sum, $memo[1]+$entry[1]] }    # 将 hash 类型中的值进行累加输出
#=> [sum, 6]
```

7. inline_template

inline_template 函数可以在资源参数中嵌入 Ruby 代码。以下为代码片段。

```
cron { 'Puppet_agent':
  command => 'puppet agent --server puppet.example.com --test ',
  hour => '0',
   minute => inline_template("<%= (hostname).hash.abs %60 %> "),    # 通过 inline_template 函数在 cron 资源中增加 Ruby 的代码
}
```

通常我们通过 cron 资源来管理 Agent 访问 Master，当需要维护的 Agent 机器比较多又都同时来访问一台 Master 服务器时，就会导致 Master 负载比较高，甚至拒绝提供服务。为了减轻 Master 的负担，最好将 Agent 命令通过 cron 资源做分时处理，这里并不是直接让 cron 资源来做分时处理，而是通过在 inline_template 函数中嵌入 Ruby 代码的方式来实现分时处理。Ruby 代码（(hostname).hash.abs %60）根据 Hostname 来计算一组 hash 值，为了得到 0 ～ 59 之间的整数，这里以 %（模除）方式来限定最终取值的范围，通过取一个随机值来实现 cron 资源的分时处理。为了演示结果，笔者将 inline_template 函数使用单独写入以下测试文件中。

```
# inlin_template.pp
$random = inline_template("<%= (hostname).hash.abs % 60 %>")
```

```
notify{"$randmo":}
```

通过 puppet apply inlin_template.pp 执行，结果如下：

```
notice: 28    # 根据 hosname 计算生成随机值
notice: /Stage[main]//Notify[28]/message: defined 'message' as '28'
notice: Finished catalog run in 0.02 seconds
```

将此值 28 设置为 cron 资源中的 minute 属性，每次 Agent 访问 Master 的时间就设定在了每小时的第 28 分钟，由于每台 Agent 的 Hostname 不一致，所以最终通过 Hostname 生成的随机数也不一样，最终达到了 Agent 分时访问 Master 的目的。

8. shellquote

shellquot 函数可以自动为字符串加引号，合并串联字符串变量。以下为代码片段。

```
$source = 'source'
$target = 'target'
$string = shellquote( $source, $target )      # 通过 shellquote 函数串联 $srouce 与 $target 变量值
$command = "/bin/mv ${string}"
notify { $command: }
```

以下为运行结果。

```
notice: /bin/mv "source" "target"
```

6.4.2 其他系统函数

除了上边介绍的系统函数外，本节再来简单介绍几个其他的系统函数。

- 常用：另外常用的系统函数还包含 realize（实例化虚拟资源，在第 7 章详细介绍）、regsubst（字符串替换函数）、require（衔接类与类之间的关系函数）、include（加载函数，在 6.7.2 节介绍）。
- 日志级别：alert、crit、emerg、notice、warning、debug、err 和 info，主要用于 Puppet 的日志级别。
- 数字签名：md5 和 sha1 分别表示不同算法的数字签名。
- hiera：hiera_array、hiera_hash 和 hiera_include。
- 其他：each（返回序列中第一个参数）、split（字符串切割）、sprintf（字符串格式化）、contain（类包含）、create_resources（转换 hash，并将资源放入 catalog）、extlookup（外部文件读取）、fail（解析错误，主要用于调试代码）、file（输出文件内容函数）、filter（hash 或数组遍历函数）、fqdn_rand（生成一个 0 到最大值的随机数，如 fqdn_rand(30)）、内嵌 lookup（查找绑定函数）、map（遍历数组）、search（搜索类路径）、slice（遍历数组范围，如 slice($[1，2，3，4，5，6]，2) # produces [[1，2]，[3，4]，[5，6]]）。

- 废弃：collect（目前已更名为 map）、select（目前已更名为 filter）。

6.5 Puppet tag

Tag（中文译为“标签”，下称标签）由字母（[a ～ z][A ～ Z]）、数字（[0 ～ 9]）和下划线（_）组成。tag 的作用是给 Puppet 中的类、资源和自定义函数打标签，可以通过 puppet agent 命令，根据打的标签来独立执行某一类、资源或自定义函数。下面来看一下标签的用法。

1）如何定义 tag，以下为代码片段。

```
apache::vhost {'docs.puppetlabs.com':
   port => 80,
    tag  => ['us_mirror1', 'us_mirror2'],  # 设置此类的 tag 标签为 us_mirror1 和 us_mirror2
}
```

2）通过 puppet agent 来更新指定标签。

指定标签更新分为以下 3 步：

步骤 1　在 site.pp 文件中定义 tag。

```
file {'/etc/host':
  ensure => file,
  source => 'puppet:///modules/host',
  mode   => '0644',
  tag    => ['wds','djangowang','jack'],
}
```

步骤 2　通过 puppet master 在编译好的 catalog 中查找 wds 标识。以下为结果的截取。

```
puppet master --compile puppet.example.com | grep -10 wds
"tags": [
    "file",
    "wds",
    "class"
 ],
"type": "File",
   "file": "/etc/puppet/environments/production/manifests/site.pp"
}
```

步骤 3　在 Agent 主动更新指定的 wds 标签。

```
puppet agent -server puppet.example.com --test -tags wds
```

6.6 Puppet 关键字

Puppet 语言中的关键字即系统保留的单词或字符串，它们不能被用来声明变量、class

类、define 函数名和自定义函数等。如表 6-5 所示为 Puppet 语言的关键字。

表 6-5 Puppet 语言关键字列表

类 型	关 键 字	类 型	关 键 字
一般性关键字	case	一般性关键字	inherits
	class		node
	default		unless
	define	表达式操作符关键字	and
	else		in
	elsif		or
	if	布尔值关键字	true、false
	import	特殊值关键字	undef


注意 在 Puppet 3 版本中，保留字符也许可以作为自定义资源类型中属性的名称。相对于 2.7 版本及更早的版本的表现来说，Puppet 3 版本无疑做了重大的改变。

除此之外还有一些资源名和函数名也是保留的。更多信息请参考官方网站资源名关键字 http://docs.puppetlabs.com/references/latest/type.html 和函数名关键字 http://docs.puppetlabs.com/references/latest/function.html。

6.7 Puppet 编程规范

为什么要介绍 Puppet 编程规范？好的规范可以促进团队合作，减少 bug 处理，降低维护成本，并有助于代码审查等。所以这里有必要了解一下 Puppet 编程规范，并加以重视。下面来介绍一下 Puppet 的符号使用的规范、代码注释的风格、变量的规范、资源的规范和类的规范。

6.7.1 manifests 和 modules 中的间距、缩进与空白

Puppet 对 Agent 的配置清单主要集中在 manifests 和 modules 两个目录中，这里介绍 manifests 和 modules 目录中清单代码文件的符号间距、缩进与空白的使用规范，也就是在 manifests 和 modules 两个目录中所有以 *.pp 为结尾的文件，都需要遵守此规范。

- 建议使用两个空格的软标签。
- 不推荐使用制表符（\t）。
- 尾部不要包含空格。
- 建议单行不要超过 80 个字符的宽度。
- 在资源的属性中，通过 => 符号进行属性的对齐。

6.7.2 注释

在编写程序语言时写注释是非常好的习惯，可以让他人方便地了解程序逻辑、程序开发者是谁和程序开发时间等信息。在 Puppet 语言中也是支持注释的，它目前支持两种风格语言注释，一种是 Shell 语言注释风格，另一种是 C 语言注释风格。下面来看一下两种风格的语言注释。

1. Shell 脚本风格注释

Shell 脚本注释风格通常也是 Ruby 语言的注释风格，它以“#”作为注释开始，可以在代码中通过“#”方式来标识代码逻辑和其他的注释信息等。不过目前“#”不支持多行注释。Shell 脚本风格注释示例内容如下。

```
# this is test
file {'/etc/ntp.conf':
 ensure => file,
 owner  => 'root',
}
```

2. C 语言风格注释

C 语言注释风格以“/*”作为开始，以“*/”作为注释结束。它可支持多行注释，但不支持 C 语言的“//”单行注释。C 语言风格注释的示例内容如下。

```
/*
author: wds
time: 2013.12.12
*/
file {'/etc/ntp.conf':
 ensure => file,
 owner  => 'root',
}
```

6.7.3 变量规范

下面来介绍定义变量的规范，同时介绍如何正确使用‘’、“”与 {} 等符号。

变量只包含字母（[a ～ z][A ～ Z]）、数字（[0 ～ 9]）和下划线（_）。以下为正确的定义与错误的声明方式。

正确：

```
$foo_bar123
```

错误：

```
$foo-bar123
```

要正确地使用‘’、“”与{}等符号。在不包含变量的字符串中都应该通过‘’进行引用；如果字符串中包含变量则可以通过“”进行引用；如果字符串中即有变量又有其他字符串，可以通过{}进行引用。以下为‘’、“”与{}推荐与不推荐的书写方式。

推荐：

```
"/etc/${file}.conf"
"${::operatingsystem} is not supported by ${module_name}"
```

不推荐：

```
"/etc/$file.conf"
"$::operatingsystem is not supported by $module_name"
```

变量被引用时不用加“”。

推荐：

```
mode => $my_mode
```

不推荐：

```
mode => "$my_mode"
mode => "${my_mode}"
```

6.7.4 资源规范

关于“资源”会在第7章中详细介绍。以下为与资源相关的规范。

1. 资源的标题

资源的标题需要用单引号或双引号引起来，同时标题中不能包含空格和连字符。以下为推荐与不推荐的书写方式。

推荐：

```
package { 'openssh': ensure => present }
```

不推荐：

```
package { openssh: ensure => present }      # openssh 缺少单引号
```

2. 符号对齐方式

资源中所有的属性与值需要以=>符号为准尽量对齐。以下为推荐与不推荐的书写方式。

推荐：

```
exec { 'blah':
  path => '/usr/bin',
```

```
  cwd  => '/tmp',
}
exec { 'test':
  subscribe   => File['/etc/test'],
  refreshonly => true,
}
```

不推荐：

```
exec { 'blah':
  path => '/usr/bin',     # 没有按照 "=>" 符号对齐
       cwd => '/tmp',
}
exec { 'test':
  subscribe => File['/etc/test'],
refreshonly => true,
}
```

3. 属性的顺序

当资源中出现 ensure 属性时，建议将此属性写在资源首部。

```
file { '/tmp/readme.txt':
 ensure => file,            # 建议 ensure 写在首部
 owner  => '0',
 group  => '0',
 mode   => '0644',
}
```

4. 资源关联关系

资源应该由逻辑关系被划为一组，而非通过资源类型来划分。以下为正确的描述方式和错误的描述方式。

正确：

```
file { '/tmp/a':
 content => 'a',
}
exec { 'change contents of a':
 command => 'sed -i.bak s/a/A/g /tmp/a',
}
file { '/tmp/b':
 content => 'b',
}
exec { 'change contents of b':
 command => 'sed -i.bak s/b/B/g /tmp/b',
}
```

错误：

```
file {     # file 中包含了两次写文件逻辑
 "/tmp/a":
  content => "a";
 "/tmp/b":
  content => "b";
}
exec {     # exec 中包含了两次替换文件逻辑
 "change contents of a":
  command => "sed -i.bak s/b/B/g /tmp/a";
 "change contents of b":
 command => "sed -i.bak s/b/B/g /tmp/b";
}
```

5. 软链接

通过 file 资源建立软链接的时候，需要设置 ensure=>link，并通过 target 属性来指定目标文件。以下为正确的书写形式与错误的书写形式。

正确：

```
file { '/var/log/syslog':                # 源文件
  ensure => link,     # 创建软链属性
  target => '/var/log/messages',     # 目标文件
}
```

错误：

```
file { '/var/log/syslog':
  ensure => '/var/log/messages',
}
```

6. 文件模式

通过 file 资源设置文件权限时需要注意以下两点：

- 文件权限应该由 4 位数字组成，而非 3 位。
- 权限数字应该通过 '' 引用。

推荐：

```
file { '/var/log/syslog':
  ensure => present,
  mode   => '0644',
}
```

不推荐：

```
file { '/var/log/syslog':
  ensure => present,
  mode   => 644,
}
```

7. 资源默认属性

默认资源可以为资源提前声明属性和值，这就解决了定义相同资源属性重定义的情况。这里需要注意默认资源需要声明在 top 域中，推荐写在 site.pp 文件开始位置。以下为推荐书写形式与不推荐书写形式。

推荐：

```
# /etc/puppetlabs/puppet/manifests/site.pp:  # 推荐将定义资源默认属性放到 site.pp 文件的首部
File {
  mode  => '0644',
  owner => 'root',
  group => 'root',
}
```

不推荐：

```
# /etc/puppetlabs/puppet/modules/ssh/manifests/init.pp  # 不推荐定义在 init.pp 文件中
File {
  mode  => '0600',
  owner => 'nobody',
  group => 'nogroup',
}
class {'ssh::client':
  ensure => present,
}
```

6.7.5 条件语句规范

通常不建议将 selector 语句与资源混用。以下为推荐与不推荐的书写形式。

推荐：

```
$file_mode = $::operatingsystem ? {
  debian => '0007',
  redhat => '0776',
  fedora => '0007',
}
file { '/tmp/readme.txt':
  content => "Hello World\n",
  mode    => $file_mode,
}
```

不推荐：

```
file { '/tmp/readme.txt':
  mode => $::operatingsystem ? {
    debian => '0777',
    redhat => '0776',
   fedora => '0007',
   }
}
```

6.7.6 class 类规范

在第 5 章中已经介绍过“类”了，下面专门来介绍一下 class 类的相关的规范。

1. 独立文件

为了代码与逻辑的清晰，所有的资源与类需要定义在单独的文件中，并通过 init.pp 文件来组合这些文件、资源与类的关系。下面以代码片段的方式来介绍文件之间的关联关系。

```
# /etc/puppetlabs/puppet/modules/apache/manifests
# init.pp
class apache {
  include::apache::ssl
  include::apache::virtual_host
}
# ssl.pp
class apache::ssl {…}
# virtual_host.pp
define apache::virtual_host () {…}
```

下面简单分析一下以上代码片段的关联关系。

- ❑ init.pp 文件：建立文件之间的关联关系。
- ❑ ssl.pp 文件：安装 Apache 的 SSL 相关环境。
- ❑ virtual_host.pp 文件：定义虚拟主机。

2. 类的内部组织结构

以下为类内部组织结构的要求：

- ❑ 需要定义类与参数。
- ❑ 需要对参数进行验证。
- ❑ 通过条件语句组织内部类关系。
- ❑ 提前预设参数。
- ❑ 声明局部变量。
- ❑ 建立资源与资源之间的关联关系。

❑ 声明默认资源，并覆盖资源属性。

类的范例：

```
# 定义类名与参数
class myservice($ensure='running') {
# 对输入的参数进行合法校验
if $ensure in [ running, stopped ] {
    $_ensure = $ensure
} else {
       fail('ensure parameter must be running or stopped')
}
# 通过条件语句组织类内部关系
case $::operatingsystem {
  centos: { $package_list = 'openssh-server'}     # 预设参数
  solaris: {$package_list = [ SUNWsshr, SUNWsshu ]}     # 预设参数
  default: {fail("Module ${module_name} does not support ${::operatingsystem}")  # 预设参数
}
# 声明局部变量 $variable = 'something'
# 声明默认资源并覆盖资源属性 Package { ensure => present, }
File { owner => '0', group => '0', mode => '0644' }
package { $package_list: }
file { "/tmp/${variable}":
  ensure => present,
}
service { 'myservice':
  ensure     => $_ensure,
hasstatus => true,}
 }
```

（1）符号关联关系

通过 -> 符号建立资源之间的关联关系的顺序为从“左到右”。

正确：

```
Package['httpd'] -> Service['httpd']
```

错误：

```
Service['httpd'] <- Package['httpd']
```

（2）模块的继承

继承可以在模块中使用，但不推荐跨模块的命名空间使用。

推荐：

```
class ssh { ... }
class ssh::client inherits ssh { ... }
class ssh::server inherits ssh { ... }
class ssh::server::solaris inherits ssh::server { ... }
```

跨模块继承。不推荐：

```
class ssh inherits server { ... }
class ssh::client inherits workstation { ... }
class wordpress inherits apache { ... }
```

6.7.7 标识符命名规范

1）变量命名规则：

- 符合正则表达式规范（\A\$[a-zA-Z0-9_]+\Z），不包含特殊的字符，如 %@^ 等。
- 变量名区分大小写，如 $foo 与 $FOO 为不同的变量。

2）class 类命名规则：

- 符合正则表达式规范（\A[a-z][a-z0-9_]*\Z），不包含特殊的字符，如 %@^ 等。
- 如果类名中使用了命名空间需要以“::”作为分隔，并符合正则表达式规范（\A([a-z][a-z0-9_]*)?(::[a-z][a-z0-9_]*)*\Z）。

3）modules 命名规则：

- 符合正则表达式规范（\A[a-z][a-z0-9_]*\Z）。
- 模块名的首字母不能为大写。

4）tag 命名规则：符合正则表达式规范（/\A[a-z0-9._-]+\Z/.）。

5）nodes 节点命名规则：符合正则表达式规范（/\A[a-z0-9._-]+\Z/.）。

6.8 Puppet 文件的导入、命名空间与自动加载

6.8.1 Puppet 文件的导入

Puppet 可以将一个文件内容插入另一个文件中，或者将多个文件内容插入一个文件中，这一功能叫做导入。manifests 和 modules 两个目录的导入方式是不一样的，下面分别看一下这两个目录的导入方式。

1. manifests 目录导入方式

在 manifests 目录内文件与文件之间的导入功能通常使用 import 函数来完成，来看一下 Puppet 文件导入的具体情况。

```
# 在 site.pp 文件中加载当前 nodes 目录下所有的 *.pp 文件
import 'nodes/*.pp'
# 在 site.pp 文件中加载 nodes.pp 文件内容
import 'nodes.pp'
```

import 函数可以导入 manifests 目录中的一个文件，也可以导入多个文件，多个文件可以使用通配符“*”来表示。

下面来看一下导入功能的使用场景。manifests 目录是一个清单文件的目录，包含任务清单，同时也包含任务的业务逻辑。如通过 Puppet 管理一个网站的多个子功能，就可以将子功能的代码独立到一个 .pp 文件中，这样写的好处是可以让代码与代码之间变得更简洁。来看以下的代码片段。

```
# site.pp 文件
import 'nodes/common.pp'
import 'nodes/web.pp'
import 'nodes/cache.pp'
import 'nodes/db.pp'
```

在 site.pp 文件中依次加载 common.pp、web.pp、cache.pp 和 db.pp 文件。其中 common 文件内容包含了网站的所有公有库文件，web.pp 文件包含了网站的所有 web 站点管理，cache.pp 文件包含了网站所有的 cache 数据，db.pp 文件包含了网站的所有数据库站点管理配置功能。这样就通过 import 函数串联了整个网站的子功能。

2. modules 目录导入方式

在第 4 章节我们介绍过 modules 基础模块目录，它类似一个仓库，存放了配置管理的模块代码和配置文件等。通常 site.pp 文件导入 modules 基础模块中的模块文件时，使用的是 include 函数，include 函数会根据 puppet.conf 文件中的 modulepath 参数搜索基础模块的模块目录路径，并自动加载基础模块目录中的 init.pp 文件。以下是在 site.pp 文件中加载 ntp 模块的代码片段。

```
#site.pp
node 'puppet.example.com' {
# /etc/puppet/manifests/modules/ntp/init.pp
  include ntp
}
```

在 site.pp 文件中通过 include 函数导入 ntp 模块，ntp 的相关配置存放在 /etc/puppet/manifests/modules/ntp/ 目录中。

6.8.2 Puppet 命名空间与自动装载

1. 命名空间

什么是命名空间？命名空间是用来组织和重用代码的编译单元。如同名字一样的意思，之所以会有 NameSpace（命名空间），是因为人类可用的单词数太少，并且不同的人写的程序不可能所有的变量都不重名。对于库来说，这个问题尤其严重，如果两个人写的库文件

中出现同名的变量或函数（不可避免），使用起来就有问题了。为了解决这个问题，引入了命名空间这个概念。Puppet 中通过“::”支持命名空间的使用方式，如以下代码。

```
class apache { ... }     # 基础类
class apache::mod { ... }     # 模块类
class apache::mod::passenger { ... }     # 模块类，表示 mod 模块下的 passenger 模块
define apache::vhost { ... }     # 自定义函数
```

如以上代码所示，apache 为基础类，通过“::”来创建模块类和自定义函数名。基础类的功能是用来确认 Apache 的状态，并能正常地运行 Apache，而模块类用于 Apache 的一些模块化的个性配置。Puppet 通过命名空间的方式来串联基础类、模块类和自定义函数等，让我们一目了然地了解模块之间的关系与作用如以下代码段：

```
class apache {
   include apache::mod
   include apache::mod::passenger
   include apache::vhost
}
```

2. 自动装载

继续刚刚的命名空间案例，我们可以直接加载 apache 的基础类和模块类。之所以可以直接写 apache 类名，就是因为 Puppet 中引入了自动装载技术。自动装载技术可以让我们省去很多的重复劳动，让代码更加整洁。通过以下代码片段，来看一下 Puppet 是如何实现自动装载的。

```
#/etc/puppet/manifests/site.ppnode default{
   include apache
}
```

其中 apache 类在 modules 基础模块目录结构如表 6-6 所示。

表 6-6 Apache 基础模块目录结构

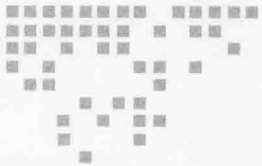
类 名	基础模块路径	文 件 作 用
apache	/etc/puppet/modules/apache/manifests/init.pp	安装逻辑
apache::mod	/etc/puppet/modules /apache/manifests/mod.pp	安装模块
apache::mod::passenger	/etc/puppet/modules/apache/manifests/mod/passenger.pp	安装模块

当 Agent 访问 Master 时会自动装载 site.pp 文件，site.pp 通过 include 函数直接装载 modules 基础模块中的 apache 类，其中 include 函数装载 modules 基础模块过程是通过 puppet.conf 中的 modulepath 参数的路径 /etc/puppet/modules/ 来完成的。根据 modulepath 参数中的路径可找到 /etc/puppet/modules/apache/manifests/ 路径，并装载 init.pp 文件中的 apache 类。init.pp 在每个 modules 模块目录中都有，它是一个初始化加载的文件，如果没

有会 Puppet 会报错。

```
#/etc/puppet/modules/apache/manifests/init.pp
class apache{
  include apache::mod
  include apache::passenger
}
```

apache 类会装载 apache::mod 和 apache::passenger 类，其中 apache::mod 为 init.pp 文件同路径下的 mod.pp 文件，主要用于安装 Apache 的相关模块；apache::passenger 为 init.pp 文件同级 mod 目录中的 passenger.pp 文件，主要用于安装 passenger 模块。site.pp 文件会自动装载 apache 目录中的 init.pp 文件，init.pp 文件会自动装载 mod.pp 和 passenger.pp 文件。3 个文件组成了对 Apache 的配置管理功能，这就是 Puppet 的自动装载功能。



Chapter 7 第 7 章

Puppet 资源详解

如果将 Puppet 语言看成 Puppet 的骨架，那么资源就是 Puppet 的精髓，整个 Puppet 的配置与管理工作都是围绕着资源来进行的，所以读者要重视对本章内容的学习。Puppet 将配置与管理操作系统的每一个功能并将其封装成为一个资源，而每个资源又由提供者兼容了不同操作系统发行版本的管理，让系统管理员不必关心底层操作系统的发行版本，而专心于配置与逻辑的管理。不仅如此，资源还提供了其他的特色功能，如虚拟资源、资源导出、默认资源等，让我们在不同的环境与需求下都能够很好地使用 Puppet 的资源功能。

本章首先介绍 Puppet 资源，让大家对资源有个基本的认识，并找到学习的方向；然后介绍 Puppet 的常用资源，并深入介绍常用资源的属性与案例，让读者对资源运用有一个初步了解与认识，为学习更多的资源打下基础；然后介绍资源的公有属性，公有属性的作用是让资源与资源之间建立关系并协调使用；接着介绍默认资源，它的作用是用来初始化资源的属性与值；最后介绍资源的高级功能——虚拟资源与资源的导出，虚拟资源主要解决资源的重定义问题，它的优势是可以提高 Puppet 代码的复用率，降低重复开发的成本；资源导出功能主要的用途是为服务器交换信息搭建一个桥梁。

7.1 Puppet 资源

在本节中，首先介绍 Puppet 资源的分类；然后介绍资源与 Puppet 的协同工作；最后介绍资源的构成。

7.1.1 Puppet 资源分类

Puppet 中的 resources（中文译为“资源”，下称资源）可以说是整个 Puppet 配置管理工具中的核心，在第 5 章，通过 Puppet 构建主机中就可以经常看到它的身影，它是通过 Puppet 管理配置系统的最小单位。截至本书出版前，Puppet 官方网站共提供了 48 个资源。笔者将它们按功能与使用频率划分为 3 类，分别是常用资源、次常用资源和 nagios 资源。

- 常用资源：file、filebucket、group、exec、host、notify、cron、service、user 和 package。
- 次常用资源：augeas、computer、interface、mailalias、maillist、mcx、router、schedule、scheduled_task、ssh_authorized_key、sshkey、zpool、yumrepo、zfs、zone、macauthorization、resources、selmodule、k5login、mount、selboolean、stage、tidy 和 vlan。
- nagios 资源：nagios_hostescalation、nagios_hostextinfo、nagios_hostgroup、nagios_serviceescalation、nagios_serviceextinfo、nagios_servicegroup、nagios_hostdependency、nagios_servicedependency、nagios_service、nagios_timeperiod、nagios_command、nagios_contact、nagios_contactgroup 和 nagios_host。

本章主要介绍常用资源，次常用资源会在后续章节涉及时单独介绍，而 nagios 资源主要管理 Nagios 监控工具，是一款开源免费的监控工具，本书并不涉及 Nagios 监控，所以这里不赘述。

7.1.2 资源与 Puppet 协同工作

在 Puppet 中每一个资源都是一种管理功能的集合，它们有着相对独立的属性和值。这些属性和值的自由组合就形成了强大的配置管理操作系统的功能。运维工程师不必关心操作系统的类型和发行版本，因为所有的资源包含一个特殊的属性——“提供者”，提供者属性已经对常用的系统发行版本进行了兼容，通常提供者可以自动判断系统发行版本，并按照系统发行版本执行相应的逻辑功能。最终 Puppet 语言与资源的结合就对操作系统实现了透明管理的功能，如图 7-1 所示。

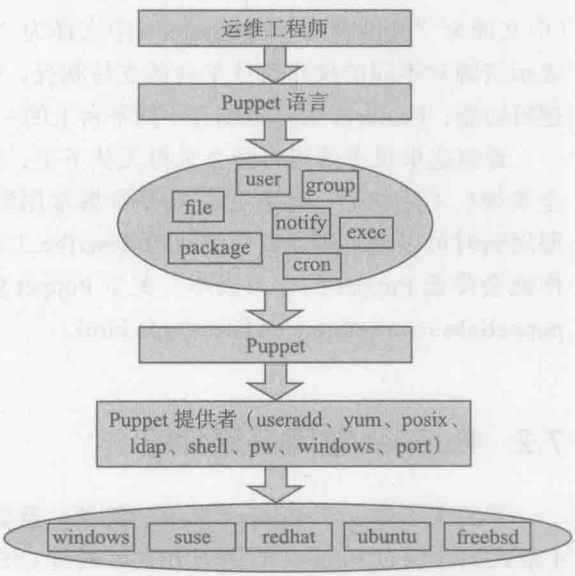


图 7-1 Puppet 资源

7.1.3 资源的组成

以下为 Puppet 资源使用的基本格式和案例。

资源的基本格式如下：

```
资源名 {' 标题 ':
 属性 1 => ' 值 ',
 属性 2 => ' 值 ',
}
```

下面以安装 Nginx 为例来介绍资源的基本组成。

资源的案例如下：

```
package {'nginx':
  ensure => present,
  provider => 'rpm',
}
```

package 资源主要用于程序包的安装。我们可以看到 package 资源中使用了 key=> values 的格式，案例介绍了通过 package 资源来安装 Nginx 这款 Web 服务器的方法，其中 nginx 标题表示安装软件的包名，ensure 属性表示安装的状态（通过更改或增 ensure 属性的值来实现对 Nginx 服务的安装、升级和删除，让软件变更变得更加灵活自如）；provider 属性为提供者，它表示安装软件使用的平台。

在学习 Puppet 资源过程中读者首先需要对资源有一个基本的了解，了解每个资源的作用和应用场景，然后再了解资源中的常用属性与用途。读者还要特别关注资源的 Providers（中文译为“提供者”）与 Features（中文译为“特性”）两个重要的属性。在这里 Providers 表示资源对不同的操作系统平台的支持状况，也就是根据操作系统的发行版本执行不同的逻辑功能；Features 表示资源在不同平台上的一些特性。

看到这里很多读者可能会觉得无从下手，这么多的资源、属性和特性什么时候才能完全掌握！不用担心，笔者建议只要掌握常用资源和它们的常用属性就可以了，其他的资源用到时可以通过第 4 章介绍过的 describe 工具或者官方网站来查找它们的使用方法，这样就会降低 Puppet 的学习成本。关于 Puppet 资源的更多信息请参考官方网站 http://docs.puppetlabs.com/references/latest/type.html。

7.2 Puppet 常用资源介绍

下面来了解一下 Puppet 常用的资源，及资源的属性、特性和资源的使用案例。在第 4 章我们介绍过 Puppet 的访问方式可通过 C/S 结构使用也可通过单机独立使用，本节主要介绍 Puppet 资源的使用和案例，对 Puppet 访问方式并不做重点讲解。所以为了降低读者的学习成本，如果不做特别介绍，本书默认采用单机独立的形式。下面依次介绍 file、filebucket、host、user、group、package、service、exec、cron 和 notify 资源常用属性和案例。

7.2.1 file 与 filebucket 资源

在很多场景中 file 与 filebucket 资源会一起使用，所以将它们放在一起来介绍。首先介绍 file 资源使用与案例，然后介绍 filebucket 资源。file 资源主要用来管理操作系统的文件、文件权限、创建目录和软连接等，它还可以通过 Puppet 的文件传输协议，从 Master 同步配置文件到 Agent 上。下面来看一下它的属性和案例。

1. file 资源常用属性及案例

以下为 file 资源的常用属性及其介绍。

```
file { '资源标题':
  path
  ensure
  backup
  checksum
  content
  force
  group
  links
  mode
  owner
  source
  target
  selinux_ignore_default
  selrange
  selrole
  seltype
  seluser
}
```

下面简单分析一下上面属性的作用。

- path：指定要管理的文件或目录的路径，必须用引号引起来（通常 path 也等于资源的标题）。
- ensure：可以分别设置 5 个值，即 absent、present、file、directory 和 link。设置 present 值表示匹配文件，它会检查 path 值中的路径文件是否存在，如果不存在就会创建新的文件；设置 absent 值表示删除 path 值中已经存在的文件；设置 file 表示创建文件；设置 directory 表示创建目录。但如果删除的是目录的话，操作上比较特殊，需要额外增加 force => true 属性和值；设置 link 值会根据 path 路径创建文件的软连接，通过 target 属性后接软连接的路径。

```
# 删除 abc 目录
file {'/tmp/abc':
  ensure => absent,
  force => true,
}
```

- backup：决定文件的内容在被修改前是否进行备份。目前 Puppet 支持两种备份方式，一种方式是将文件备份在 Agent 上被修改文件的目录中，另一种方式是将源文件通过 filebucket 备份在远程服务器上。首先看备份在 Agent 上的方式，backup 属性的值如果是以“.”开头的字符串的话，Puppet 会将变更文件备份在 Agent 源文件的同一目录下，备份文件的扩展名就是“.”值里面的字符串。如果备份在远程服务器上，需要借助 filebucket 资源，稍后会以案例进行介绍。如果不想做备份可以设置 backup => false 值。
- checksum：检查文件内容是否被修改过，通过它可以检查文件的一致性。有几种检查方式，包括 md5、mtime 和 ctime 等，默认用 md5 进行检查。
- content：可以向文件中追加内容或者通过调用 template 函数向 ERB 模板中追加内容。ERB 模板会在第 8 章详细介绍。

```
# 将 "nameserver:192.168.1.1" 追加到 /etc/resolve.conf 文件
file {'/etc/resolv.conf':
 content => 'nameserver: 192.168.1.1',
}
```

- force：可以将一个目录变成一个链接，可用的值是 ture、false、yes 和 no，其中 true 与 yes 参数在这里均表示创建目录链接，false 与 no 参数均表示不创建目录链接。类似的同一参数相同值的情况曾在第 4 章介绍过，这里就不再介绍。
- group：可以指定该文件的用户组，值可以是组的 gid 或系统组名。
- links：定义操作符合链接的文件，可以设置的值是 follow 和 manage。设置 follow 值，文件复制时，会复制文件的内容，而不是只复制符合链接本身；如果设置成 manage 值，则会复制符合链接本身。
- mode：用于设置文件的权限。

```
# 修改文件权限，将 passwd 值设置为 644
file {'/etc/passwd':
  mode =>'644',
}
```

- owner：设置文件的属主。
- source：指定源文件的位置，值可以是指定的远程文件的 URIS 或者本地的完整路径。

```
# 通过 Puppet 数组同步多文件
file {'/etc/nfs.conf':
  source => [
    "puppet:///modules/nfs/conf.$host",
    "puppet:///modules/nfs/conf.$operatingsystem",
    "puppet:///modules/nfs/conf"
  ]
}
```

- target：指定创建软连接的目标。

```
file {'/etc/inetd.conf':
ensure => link,
target => 'inet/inetd.conf',
}
```

- selinux_ignore_default：SELinux 系列功能，实现自定义 SELinux。
- selrange：SELinux 系列功能，定义范围。
- selrole：SELinux 系列功能，定义角色。
- seltype：SELinux 系列功能，定义类型。
- seluser：SELinux 系列功能，定义用户。

下面将介绍两个与 file 资源相关的案例，第一个案例是通过 file 资源修改系统文件权限和属主，第二个案例是通过 file 资源从 Master 上同步配置文件到 Agent 上的指定位置。

案例 1

通过 file 资源修改系统上的 /etc/passwd 和 /etc/shadow 文件属性，设置它们的 owner 和 group 的文件权限。编辑 /etc/puppet/manifests/file.pp 文件，将以下内容追加到文件中。

```
file {'/etc/passwd':
 owner => 'root',
 group => 'root',
 mode  => '644',
 }
file {'/etc/shadow':
 owner => 'root',
 group => 'root',
 mode  => '440',
}
```

通过 file 资源首先设置 /etc/passwd 文件权限，通过设置标题为“/etc/passwd”的 file 资源指定要配置权限文件的位置；通过设置 mode 属性指定文件权限为 644，也就是设置系统文件的 user、group 和 other 权限，6 表示 user（可读 / 执行）、group 和 other 用户可读；设置它们的属主分别为 root。然后通过 file 资源设置 /etc/shadow 文件权限，设置 file 资源标题为“/etc/shadow”，设置 mode 属性指定文件权限为 440；设置它们的属主分别为 root。设置完成后通过 puppet apply 命令来进行测试，过程和结果如下：

```
# puppet apply  /etc/puppet/manifests/file.pp
notice: /Stage[main]//File[/etc/shadow]/mode: mode changed '644' to '440'
notice: /Stage[main]//File[/etc/passwd]/mode: mode changed '700' to '644'
notice: Finished catalog run in 0.01 seconds
```

执行后 Puppet 会将执行结果显示在终端上，并告诉我们按照 /etc/puppet/manifests/file.pp 文件进行的配置，都对系统做了哪些变更。从输出结果上可以看到，文件的权限都有了变更，细心的读者可能会发现文件的 owner 和 group 没有变化。为什么呢？其实是这样的，

Puppet 在更改系统文件属性前，会对照配置文件与目标文件属性的差异，并对差异部分进行更改，因为修改前的 /etc/passwd 和 /etc/shadow 文件属性就是 root，所以没有再次更改。

案例 2

此案例为 C/S 结构，在 Master 上将 Apache 的 httpd.conf 配置文件同步到 Agent 上。在同步过程中如果 Master 上的原文件与 Agent 上的目标文件不一致，需要对源文件进行备份后再进行同步覆盖。这里的备份方式在刚刚介绍 file 资源的 backup 属性的时候也介绍了。这里首先介绍将源文件备份在 Agent 上的方式，然后介绍将源文件备份在 Master 上的方式。在 Master 上编辑 /etc/puppet/manifests/site.pp 文件，将以下内容追加到文件中。

```
node default{
  file {'/etc/httpd/conf/httpd.conf':     # Agent 目标位置
    mode => '777',owner => 'httpd',group => 'httpd',
    backup => '.bak',
    source => "puppet: ///files/httpd/httpd.conf",     # Master 源文件位置
  }
}
```

首先定义 file 资源标题，指定 httpd.conf 配置文件在 Agent 上需要同步的目标位置。然后设置它的 mode 属性为 777，即 user、group 和 other（可读、可写、可执行）；设置 owner 和 group 属性均为 httpd；设置 backup 属性 “.”，表示文件同步文件时进行 Agent 本地备份，bak 为备份文件的扩展名；最后 source 属性指定 httpd.conf 配置文件在 Master 服务器上的位置。其中 puppet:/// 的挂载路径由 Master 上的 fileserver.conf 文件指定，如图 7-2 所示。其使用了 Puppet 自带的文件传输协议。

```
[redacted]:~ # cat /etc/puppet/fileserver.conf
[files]
path /etc/puppet/files
allow *
[redacted]:~ # ls /etc/puppet/files/httpd/httpd.conf
/etc/puppet/files/httpd/httpd.conf
```

图 7-2 fileserver.conf 配置文件

设置完成后，返回 Agent 服务器，通过 puppet agent 命令在 Agent 上来测试它。过程和结果如下：

```
# puppet agent --server puppet.example.com --test
info: Caching catalog for puppet_agent
info: Applying configuration version '1383567132'
--- /etc/httpd/conf/httpd.conf  2011-11-04 20: 09: 56.000000000 +0800
+++ /tmp/puppet-file20131104-5353-f4gmlx-0      2011-11-04 20: 12: 13.000000000 +0800
@@ -1,4 +1,4 @@
-#     变更前内容
+#     变更后内容
 # Based upon the NCSA server configuration files originally by Rob McCool.
 #
 # This is the main Apache server configuration file.  It contains the
```

```
   err: /Stage[main]//Node[default]/File[/etc/httpd/conf/httpd.conf]/content: change
from {md5}9a245c811b297e11dfc8981318a456f7 to {md5}45bce543a9153016a6aef2d5cb5324be
failed: Could not find group httpd
   notice: Finished catalog run in 1.32 seconds
```

Puppet 同步配置文件时，如果配置文件有变更，会输出整个配置文件的原内容和变更内容，由于篇幅有限，笔者进行了一定的删减，只保留了重要的变更部分。为了演示结果，笔者在同步前修改了 Master 上的 httpd.conf 配置文件内容，去掉了配置文件行首的一个“#”，从输出的结果可以看出，如果 httpd.conf 配置文件有变更，Puppet 会在开始显示出变更部分的内容，让我们一目了然它本次变更了什么。同时根据我们资源配置的 backup 属性，Puppet 在同步配置文件后，会将变更前的 httpd.conf 配置文件在 Agent 的 /etc/httpd/conf/ 目录下进行备份，备份文件名就是 httpd.conf.bak。

2. filebucket 资源及案例

filebucket 资源主要用于文件的备份与恢复，它通常与 file 资源配合使用。先来看一下 filebucket 资源的属性，再来看它的案例。

以下为 filebucket 资源的属性。

```
filebucket {' 资源标题 '
 name
 path
 port
 server
}
```

下面简单分析一下上面属性的作用。

- name：filebucket 的名字。
- path：服务器备份数据路径。
- port：备份服务器的端口。
- server：备份服务的域名。

这里继续 file 资源的案例，备份 httpd.conf 配置文件到远程 Master 服务器上。再次编辑 /etc/puppet/manifests/site.pp 文件，将以下内容追加到文件中。

```
node default{
  filebucket {'main':
   server =>'puppet.example.com',
   path   =>'/var/lib/puppet/clientbucket/';
}
  file {'/etc/httpd/conf/httpd.conf':
   mode =>'777',owner =>'httpd', group =>'httpd',
   backup =>'main',
   source =>'puppet: ///files/httpd/httpd.conf',
  }
}
```

在 filebucket 资源中 main 为资源的标题；server 属性是远程备份服务器域名；path 属性是远程服务器备份文件目录。在刚刚的 file 资源的基础上，我们只要修改 backup 属性即添加 filebucket 资源的 main 标题就可以了。然后再次回到 Agent 执行 puppet agent 命令，如果 httpd.conf 配置文件有变更，Puppet 就会在 Master 指定目录中将源文件备份一份。

7.2.2 host 资源

host 资源主要用来管理操作系统的 hosts 功能，hosts 是一个没有扩展名的系统文件，基本作用就是将一些常用的域名与其对应的 IP 地址建立一个关联“数据库”。当用户在浏览器中输入一个需要登录的网址时，系统会首先自动从 hosts 文件中寻找对应的 IP 地址，一旦找到，系统会立即打开对应网页；如果没有找到，系统会将网址提交到 DNS 域名解析服务器进行 IP 地址的解析。在 UNIX/Linux 系列系统中，大多数默认 hosts 功能的配置文件都存放在 /etc/hosts 文件中，但部分系统的 hosts 功能配置文件也有差异，如苹果系列的系统对于这种操作系统有不同的解决方案，这里不做过多介绍。下面来看一下 host 资源的属性和案例。

1. host 资源常用属性

以下为 host 资源的常用属性。

```
host { '资源标题':
host_aliases
ensure
ip
name
qtarget
}
```

下面简单分析一下上面属性的作用。

- host_aliases：主机能有任意别名。多个值需要指定为一个数组。

```
# 创建 puppet.example.com 的别名 ["web","db"]
host { 'puppet.example.com':
  ip => '192.168.7.7',
  host_aliases => [ "web", "db" ],
  ensure => present,
}
```

- ensure：确定该主机是否启用，值为 present 即启用，值为 absent 即关闭。
- ip：主机的 IP 地址，目前支持 IPv4 和 IPv6 两个版本。
- name：主机名。
- target：指定自定义 host 文件的位置。

2. host 资源案例

通过 host 资源设置 UNIX/Linux 系统的 hosts 功能，编辑 /etc/puppet/manifests/hosts.pp 文件，将以下内容追加到文件中。

```
host { 'test.host.com':
  ensure => present,
  ip => '192.168.1.1',
}
```

设置 host 资源标题指定测试域名为 test.host.com；设置 ip 属性为 192.168.1.1，此 IP 为标准的 IPv4 版本，也可以设置 IPv6 版本地址；设置 ensure 属性的值为 present，表示增加 hosts 功能，反之设置 absent 属性表示删除此功能。设置完成后通过 puppet apply 命令来测试它，过程和结果如下：

```
# puppet apply /etc/puppet/manifests/hosts.pp
notice: /Stage[main]//Host[test.host.com]/ensure: created
notice: Finished catalog run in 0.01 seconds
```

通过输出结果，可以看到 Puppet 已经添加了 test.host.com 域名和其对应的 IP 到系统的 hosts 功能中，这时再系统访问 test.host.com 域名时，系统会优先读取 /etc/hosts 文件，将域名解析为相应的 IP。增加完成后也可以通过 ping 命令测试域名解析结果如图 7-3 所示。或通过系统提供的 cat 命令查看 /etc/hosts 文件，来确认它是否已经被成功设置。

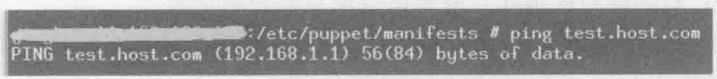


图 7-3 通过 ping 命令测试结果

注意，如果测试 test.host.com 解析失败，很有可能是 host.conf 配置问题导致。host.conf 配置文件的作用是设置 host 与 bind 的解析顺序，可以通过追加设置解析顺序的方式来解决遇到的问题。解析顺序设置如下：

```
echo " order hosts,bind " >> /etc/host.conf
```

7.2.3 user 资源

user 资源主要用来管理操作系统的账号，如账号的增加 / 删除、账号 uid/gid 的管理、shell 的解析器、宿主目录的设置和账户密码的设置等。下面来看一下它的属性和案例。

1. user 资源常用属性

以下为 user 资源的常用属性。

```
user { '资源标题':
```

```
  name
  ensure
  allowdupe
  comment
  uid
  gid
  groups
  home
  managehome
  manages_expiry
  password
  manages_password_max_age
  manages_password_min_age
  shell
  provider
}
```

下面简单分析上面属性的作用。

- name：创建的系统账户名，这里建议账户名长度小于 8，并以字母开头。
- ensure：设置账户的增加和删除，可以设置为 present 值表示增加账户，设置为 absent 值表示删除账户。
- allowdupe：是否允许系统存在同样账户的 uid，可以设置 false 表示不可以设置相同 uid，设置 true 表示可以设置相同 uid。默认值为 false。
- comment：对该账户的描述，通常可以填写账户的全名。
- uid：系统账户的 uid 必须设置成数字，如果不设置此属性系统将会自动分配账户。
- gid：系统账户的 gid 可以用数字或者组名字。
- groups：指定该系统账户属于哪些组的成员，主组不必在这里列出，多个组用数组列出，如 ['gorup1', 'group2']。
- home：系统账户的宿主目录。注意，这个目录需要提前通过 mkdir 系统命令创建。
- managehome：管理用户的时候是否管理用户的 home 目录，可以设置的值是 true、false、yes 和 no，默认值为 false。
- manage_expiry：管理用户的过期时间。
- password：系统账户的密码，具体用什么加密方式由操作系统决定，需 manages_passwords 特性，如果密码里面带有 $ 符合，需要用单引号引起来。
- manage_password_max_age：设置密码的过期期限，此属性为修改密码的最大时间。
- manage_password_min_age：设置密码的过期期限，此属性为修改密码的最小时间。
- shell：用户的 shell，即指定用户的命令解析器。
- provider：aix（AIX 系统用户管理）、dirctoryservice（Mac OS X 系统用户管理）、hpuxuseradd（HP 系统用户管理）、ldap（LDAP 用户管理）、pw（FreeBSD 用户管理）、user_role_add（Solaris 系统用户管理）、useradd（RedHat 系列系统用户管理）

和 windows_adsi（Windows 系列用户管理）。

2. user 资源案例

通过 user 资源在 UNIX/Linux 系统上增加 test 账户，并设置 uid/gid、宿主目录、账户密码和 shell 解析器。在创建系统账户前需要做一些准备，首先通过命令来生成一个加密的密码，以 RedHat 为例，系统账户默认将加密的密码存放在 /etc/shadow 文件中，可以通过系统命令 openssl passwd -1 来生成加密的密码。如果此命令不存在，可以通过 yum -y install openssl* 来安装命令所在的软件包，在系统终端键入 openssl passwd -1 命令和参数时，系统会提示输入密码，这里可以输入设置的账户密码，经过两次输入确认后，密码会以加密的形式显示在终端上。具体如下：

```
# openssl passwd -1
Password:
Retype password:
$1$d6xCR1$Lj.RK/KiP76v02wdDs25n.
```

继续编辑 /etc/puppet/manifests/user.pp 文件，将以下内容追加到文件中。

```
user {'test':
  uid =>'501',
  gid =>'501',
  home =>'/home/test',
  password =>'$1$d6xCR1$Lj.RK/KiP76v02wdDs25n.',     # openssl 命令生成的加密密码
  shell =>'/bin/bash';
}
```


password 属性的值用来设置账户的密码，通常用单引号（‘’）将加密引起来，因为值包含一些特殊符号，如果用（“”）会导致一些错误的发生。

user.pp 文件中内容如下：首先调用 user 资源创建 test 账户，设置标题为要创建的账户名；设置 uid/gid 属性分别为 501，表示 uid 号与 gid 号；设置 home 属性宿主目录为 /home/test，表示指定用户的根目录路径；设置 password 属性的账户密码为刚刚通过 openssl 命令生成后的加密密码，切记要用（‘’）；设置 shell 的解析器为 /bin/bash。

这里读者需要了解 Puppet 的工作原理，以上操作并不会真正编辑操作系统的 /etc/passwd 和 /etc/shadow 文件来增加账户和其属性，只会利用操作系统的 POSIX API（中文译为“标准的接口”，即通过系统命令）帮我们创建账户，以 RedHat 为例，会使用 useradd 命令。所以 useradd 命令并不会帮我们创建账户的宿主目录，需要在通过 user 资源创建好账户后，再次调用 file 资源来帮我们创建账户的宿主目录，这里可以将 user 资源和 file 资源写到同一个文件中来执行。创建账户宿主目录，继续编辑 /etc/puppet/manifests/user.pp 文件，将以下内容追加到文件中。

```
file {'/home/test':
ensure => directory,
 group => '501',
  owner => '501',
   mode => '700',
}
```

file 资源设置账户的宿主目录；设置 group 和 owner 属性为 501；mode 属性为 700，即 user（可读、可写、可执行），group 和 other 没有任何权限；最后设置 ensure 属性为 directory，指定它创建的为目录。设置完成后通过 puppet apply 命令来进行测试，过程和结果如下：

```
# puppet apply /etc/puppet/manifests/user.pp
notice: /Stage[main]//File[/home/test]/ensure: created
notice: Finished catalog run in 0.03 seconds
```

创建 test 账户后，可以通过查看 /etc/passwd 和 /etc/shadow 两个文件的内容，确认 test 账户是否已经被成功添加。

7.2.4 group 资源

既然有管理系统账号的资源，自然也就有管理系统组的资源。group 资源的主要功能是管理系统组，包含组的名字、增减 / 删除组、组成员、组的 gid 等。下面来看一下它的属性和案例。

1. group 资源常用属性

以下为 group 资源的常用属性。

```
group{ '资源标题':
 allowdupe
 ensure
 gid
 members
 name
 allowdupe
 provider
}
```

下面简单分析一下上面属性的作用。

- allowdupe：是否允许系统存在同样账户的 gid（此属性不能在 freebsd 上面使用），可以设置 false 表示不可以设置相同 uid；设置 true 表示可以设置相同 uid。默认值为 false。
- ensure：创建或者删除组，设置 absent 为删除组，设置 present 为创建组。

- gid：设置组的 gid，必须是数字，如果不指定，将自动分配，不同的系统自动分配的算法不一样，不推荐使用自动分配 gid。
- members：该组的成员。
- name：该用户组的名字，默认与标题相同。
- allowdupe：准许使用相同的 gid，默认为 false。
- provider：aix（AIX 系统组管理）、dirctoryservice（Mac OS X 系统组管理）、ldap（LDAP 组管理）、groupadd（RedHat 系列组管理）、pw（FreeBSD 组管理）和 windows_adsi（Windows 系列组管理）。

2. group 资源案例

通过 group 资源在 UNIX/Linux 系统上增加 test 组，增加组后再测试一下如何删除此组。编辑 /etc/puppet/manifest/group.pp 文件，将以下内容追加到文件中。

```
group {'test':
ensure => present,
    gid => '107',
}
```

通过 group 资源指定增加组的名字为 test；设置 gid 属性为 107；设置 ensure 属性为 present，表示增加组。设置完成后通过 puppet apply 命令来进行测试，过程和结果如下：

```
# puppet /etc/puppet/manifests/group.pp
notice: /Stage[main]//Group[test]/ensure: created
notice: Finished catalog run in 0.04 seconds
```

Puppet 输出结果显示，已经成功地创建了 test 组。可以查看 /etc/group 文件确认 test 组是否已经被创建成功。创建成功后再来看一下如何删除 test 组，这个很简单，只要将 ensure 属性的值改为 absent 就可以了，这里就不进行案例演示了，读者可以自己测试一下。

7.2.5 package 资源

package 资源可以借助本地包管理系统帮我们安装软件，也可以通过参数指定软件包来安装。下面来看一下它的属性、特性与案例。

1. package 资源常用属性

以下为 host 资源的常用属性。

```
package{'资源标题':
 allowcdrom
 description
 ensure
 provider
```

```
  source
}
```

下面简单分析一下上面属性的作用。

- allowcdrom：通知 apt 允许使用 cdrom 作为软件源，可以设置成 false 或者 true。
- description：描述软件包，软件包设置的一个只读属性。
- ensure：设置软件包的安装状态。installed 值或 present 值均表示已安装软件包，其最终结果是一致的；absent 值表示卸载软件包；pureged 值表示移除软件包；latest 表示安装软件包的最新版本（建议慎用此值。以 PHP 安装为例，latest 会根据 PHP 的版本安装最新的软件包，而不同的 PHP 版本之间会有一些细微的差异，不经测试直接安装最新的软件包会导致服务异常或不可用）；建议直接写安装软件的版本号方式，如 PHP 的版本号为 5.4.29，可以写为 ensure =>"5.4.29"。
- provider：不同的操作系统平台有着相对独立的包管理系统或者叫包管理器（下称包管理系统），如 RedHat 使用的是 yum 包管理系统，uBuntu 使用的是 apt 包管理系统。目前 package 资源还支持 aix、appdmg、apple、apt、dpkg、fink、freebsd、gem、hpux、macports、msi、nim、openbsd、opkg、ports、rpm、rug、windows、yum 和 zypper 等包管理系统和工具。
- source：指定软件包的安装源，如 rpm 包的 URL 地址或本地路径等。

2. package 资源特性

下面简单分析一下 package 资源的特性。

- holdable：保持现状，不自动更新软件包及依赖关系。
- install_options：个性化安装软件包，用于安装时传递相关命令。

```
package { 'mysql':
  ensure => installed,
  source => 'N:/packages/mysql-5.5.16-winx64.msi',
  install_options => [ '/S', { 'INSTALLDIR' => 'C:\mysql-5.5' } ],
}
```

- installable：安装软件包。
- purgeable：清理相关软件包，清理内容包含软件包目录配置文件等。
- uninstall_options：卸载软件包，指定卸载软件包的相关参数。

```
package { 'VMware Tools':
  ensure => absent,
  uninstall_options => [ { 'REMOVE' => 'Sync,VSS' } ],
}
```

- uninstallable：卸载软件包。
- upgradeable：升级软件包。

❑ versionable：指定软件包版本安装。

目前这 8 个特性在不同的系统平台的支持情况是不一样的，由于 package 资源的 Provider 比较多，这里只介绍常用的 Provider 对以上特性的支持状况，并排除 install_options 和 uninstall_optionst 两个特性，因为它们支持的 Provider 比较有限。特性支持状况如表 7-1 所示，其中 Y 表示支持，N 表示不支持。关于 package 资源更多的 Provider 的情况请参考官方网站 http://docs.puppetlabs.com/references/latest/type.html#package。

表 7-1 package 特性与系统平台支持状况

Provider	installable	uninstall	holdable	purgeable	Upgradeable	versionable
Yum	Y	Y	N	Y	Y	Y
Apt	Y	Y	Y	Y	Y	Y
Gem	Y	Y	N	N	Y	Y
Msi	Y	Y	N	N	N	N
Rpm	Y	Y	N	N	Y	Y
Windows	Y	Y	N	N	N	Y

3. package 资源案例

来看 package 资源的两个案例，案例 1 介绍如何通过 package 资源安装本机的 RPM 软件包，案例 2 介绍如何通过网络来批量安装软件包。

案例 1

通过 package 资源安装 Nginx 的 RPM 软件包的流程如下：首先下载 Nginx 的 RPM 软件包到指定的目录（如 /tmp/ 目录）。读者可以在网上自行找一下 Nginx 的 RPM 软件包，由于不是重点这里不作介绍。编辑 /etc/puppet/manifests/package.pp 文件，将以下内容追加到文件中。

```
package { 'nginx':
  ensure => installed,
  source => '/tmp/nginx-1.2.2-2.x86_64.rpm',
  provider => 'rpm',
}
```

设置 package 资源的标题为 nginx，如果不指定路径安装的话，package 资源会根据这一标题通过本机的包管理工具进行安装。由于这里设置了 source 属性，所以标题只起到了标识的作用；设置 ensure 属性为 installed 表示软件包的状态为安装；设置 source 属性指定 Nginx 的 RPM 软件包在本机的位置；设置 provider 属性为 RPM，也就是指定 Nginx 软件包以 RPM 的方式安装。设置完成后通过 puppet apply 命令来测试它，过程和结果如下：

```
# puppet apply /etc/puppet/manifests/package.pp
notice: /Stage[main]//Package[nginx]/ensure: created
notice: Finished catalog run in 0.52 seconds
```

通过 Puppet 输出可以看到，已经成功地安装了 Nginx 软件包。还可以通过 RPM 命令来检查 Nginx 软件是否安装成功。

```
# rpm -qa | grep nginx
nginx-1.2.2-2
```

案例 2

如果安装的软件比较多，推荐使用 Puppet 提供的数组功能来安装。如果机器可以访问互联网可以不指定 source 属性，Puppet 默认通过本机的包管理工具从互联网上安装。package 资源会从互联网依次下载并安装其内容。通过 Puppet 数组批量安装 autoconf、bison、curl、libreadline-dev、subversion 和 zlib1g-dve 软件包，编辑 /etc/puppet/manifests/package.pp 文件，将以下内容追加到文件中。

```
package { [ "autoconf",
            "libreadline-dev",
            "subversion",
            "zlib1g-dev" ]:
             ensure => installed,
}
```

Puppet 会利用系统自带的包管理系统，并根据 ensure 属性定义的安装状态，从网上进行依次下载并安装。注意，安装前应先确认网络是否可用。

7.2.6 service 资源

通过 service 资源不但可以启动、重启和关闭程序的守护进程，监控进程状态，还可以将守护进程加入自动启动中。首先来看一下它的属性和特性。

1. service 资源常用属性

以下为 service 资源的常用属性。

```
service { '资源标题':
 binary
 enable
 ensure
 hasrestart
 hasstatus
 name
 path
 pattern
 restart
 start
 status
 stop
 provider
}
```

下面简单分析一下上面属性的作用。

- binary：指定二进制程序的系统路径，用于那些不支持 init 的操作系统。如果守护进程没有自启动脚本，可以通过此属性启动服务。
- enable：指定服务在开机的时候是否启动，可以设置 true 值和 false 值。
- ensure：是否运行服务，running 值表示运行服务，stopped 值表示停止服务。
- hasrestart：指出管理脚本是否支持 restart 值，如果不支持，就用 stop 值和 start 值实现 restart 效果。可以设置的值是 true 或 false。
- hasstatus：指出管理脚本是否支持 status 值。Puppet 用 status 值来判断服务是否已经在运行，如果系统不支持 status 值，Puppet 可以利用查找运行进程列表里面是否有服务名来判断服务是否在运行。可以设置的值是 true 或 false。
- name：守护进程服务的名字，如果忘记守护进程的名字可以在 /etc/init.d/ 目录下找到他们。
- path：启动脚本的搜索路径，可以用冒号分隔多个路径，或者用数组指定。
- pattern：设置匹配进程的字符串，当服务停止时，通过进程列表来判断服务的状态，主要用于不支持 init 脚本的系统。
- restart：指定用于重启服务的脚本，否则只能手动先停止该服务再启动该服务。
- start：指定启动服务的命令，通常 init 模式的管理脚本都支持，不需要手工指定。
- status：指定 status 命令，如果不指定，就从进程列表查询该服务。
- stop：指定停止服务的脚本。
- provider：Debian 系统支持 init 模式的管理脚本，支持 enableable 与 refreshable 特性；FreeBSD 系统支持 init 模式的管理脚本，支持 enableable 与 refreshable 特性；init 标准的 init 模式支持 refreshable；RedHat 系统支持 init 模式的管理脚本，支持 enableable 和 refreshable 特性；smf solaris 支持新的服务管理框架，支持 enableable 和 refreshable 特性。

```
service {'myservice':
  provider =>'daemontools',
  path =>'/path/to/daemons',
}
```

2. service 资源特性

下面简单分析一下 service 资源的特性。

- enableable：启动或停止服务。
- refreshable：重启服务。

不同的操作系统对 service 资源特性的支持是不同的，不同的操作系统对 service 资源两个特性的支持状况如表 7-2 所示，其中 Y 表示支持，N 表示不支持。关于 service 资

源更多的Provider状况请参考官方网站http://docs.puppetlabs.com/references/latest/type.html#service。

表 7-2 service 特性与系统平台支持状况

Provider	enableable	Refreshable
daemontools	Y	Y
debian	Y	Y
freebsd	Y	Y
gentoo	Y	Y
init	N	Y
redhat	Y	Y
windows	Y	Y

3. service 资源案例

service 资源是负责机器上进程状态的资源，下面就以 vsftpd 守护进程为例来演示一下其常用的启动 / 关闭功能和开机自启动 / 关闭等功能。

（1）启动 vsftpd 守护进程

通过 service 资源启动 vsftpd 的守护进程，其中 service 的标题内容需要在 /etc/init.d/ 中存在。

```
service { 'vsftpd':
  ensure => running,
}
```

（2）关闭 vsftpd 守护进程

通过 service 资源关闭 vsftpd 进程。

```
service { 'vsftpd':
  ensure => stopped,
}
```

（3）开机自启动进程

可以编辑 UNIX/Linux 系统的启动文件，将 vsftpd 守护进程放入系统自启动文件。另外也可以通过 service 资源实现开机自启动 vsftpd 功能。

```
service { 'vsftpd':
  ensure => true,
}
```

（4）关闭开机自启动进程

关闭 vsftpd 守护进程开机自启动。

```
service { 'vsftpd':
  ensure => false,
}
```

7.2.7 exec 资源

exec 资源的功能是调用 UNIX/Linux 系统命令，完成系统管理的基础操作。下面来看一下它的属性和案例。

1. exec 资源常用属性

以下为 exec 资源的常用属性。

```
exec { '资源标题':
  command
  creates
  cwd
  environment
  group
  logoutput
  onlyif
  path
  refresh
  refreshonly
  returns
  timeout
  tries
  try_sleep
  user
  provider
}
```

下面简单分析一下上面属性的作用。

- command：指定要执行的系统命令。
- creates：此参数会创建一个临时文件，当此临时文件不存在时 exec 调用系统命令才会执行成功，防止出现同一时刻多次执行的情况。

```
exec { 'tar -xf /Volumes/nfs02/important.tar':
  cwd => '/var/tmp',
  creates => '/var/tmp/myfile',
  path => ["/usr/bin", "/usr/sbin"]
}
```

- cwd：系统命令执行的路径，如果指定的目录不存在，命令执行将会失败。
- environment：添加系统命令的附加环境变量，也可以加入自己的 path 环境变量来覆盖系统的环境变量。添加多个环境变量需要使用数组指定。
- group：执行命令运行的账户组。
- logoutput：决定是否记录输出日志信息。默认会根据 exec 资源的日志等级来记录输出信息，使用 on_failure 时只有命令执行有误的情况下才会记录输出信息。值可以为 true、false、on_failure 和任何合法的日志等级。
- onlyif：只有 onlyif 指定命令执行返回结果为 0 的时候，命令才会执行。

```
exec { 'logrotate':
  path => '/usr/bin:/usr/sbin:/bin',
  onlyif => "test `du /var/log/messages | cut -f1` -gt 100000"
}
```

- path：命令执行搜索的路径。如果没有指定 path 环境变量，系统命令需要填写完整的路径。
- refresh：刷新命令执行状态。
- refreshonly：作为一个更新机制，当依赖的对象改变时命令才会执行。当下发的 aliasses 配置文件发生变更时，exec 资源通过 subscribe 和 refreshonly 监听到依赖文件的状态，则触发 exec 资源的执行。

```
# 下发 aliases 文件
file { '/etc/aliases':
  source => 'puppet://server/module/aliases'
}
# 当 /etc/aliases 文件变化后，重建数据库
exec { 'newaliases':
  path => ["/usr/bin", "/usr/sbin"],
  subscribe => File["/etc/aliases"],
  refreshonly => true, }
```

- returns：指定预期的返回代码，如果执行的命令返回其他的代码将会出现错误。默认是 0，可以指定一个单一的值也可以指定一个包含多个值的数组。
- timeout：指定命令运行的超时时间，单位为秒，如果命令执行的时间超过了 timeout 设定的时间，就会认为命令执行失败并且会停止该命令。设置为 0 表示没有超时的限制。
- tries：命令执行重试次数，默认为 1。设置这个值之后命令会重试设置的次数直到正确的代码返回。
- try_sleep：设置命令重试的间隔时间，单位是秒。
- user：指定执行命令的账户。
- provider：目前支持 POSIX 标准、Shell 和 Windows。

2. exec 资源案例

exec 资源可以调用系统命令，对操作系统做一些基本的操作。来看以下案例，首先通过 exec 资源以 Puppet 账户身份来解压 soft.tar.gz 文件。为了测试 exec 资源的使用方法，可以通过 tar 命令自行模拟一个压缩文件，将它放置在 /etc/puppet/manifests/soft.tar.gz。编辑 /etc/puppet/manifests/exec.pp 文件，将以下内容追加到文件中。

```
exec { 'test':
  path => ["/usr/bin","/bin"],
  creates =>'/tmp/lock',
  user => 'puppet',
  group => 'puppet',
  timeout => '3',
  command => 'tar -xzf /etc/puppet/manifests/soft.tar.gz';
}
```

path 属性主要是设置系统命令的环境变量，这里也可以通过在 command 属性中调用系统绝对路径的形式忽略设置 path 属性；creates 属性设置 lock 锁文件，锁文件不存在时 exec 资源才能执行成功；user 和 group 设置命令执行时的账户身份；timeout 属性设置 exec 资源的超时时间，通常用来解决命令假死情况；command 调用系统 tar 命令接压缩 soft.tar.gz 文件。设置完成后通过 puppet apply 命令来进行测试，过程和结果如下：

```
# puppet apply /etc/puppet/manifests/exec.pp
notice: /Stage[main]//Exec[test]/returns: executed successfully
notice: Finished catalog run in 0.14 seconds
```

执行完成后可以到 /etc/puppet/manifests/ 目录确认一下 soft.tar.gz 压缩文件是否已经被解压成功。

7.2.8 cron 资源

cron 资源主要用来管理操作系统的定时任务（即 crontab）。下面来看一下它的属性和案例。

1. cron 资源常用属性

以下为 cron 资源的常用属性。

```
cron{ '资源标题':
 command
 ensure
 environment
 hour
 minute
 month
 monthday
 weekday
 name
 provider
 user
}
```

下面简单分析一下上面属性的作用。

- command：crontab 要执行的命令，由于环境变量的问题，建议调用命令时使用绝对路径，或指定 cron 资源的 enviroment 属性。
- ensure：指定该资源是否启用，可设置 present 值表示启用，设置 absent 值表示关闭。默认为 present 值。
- environment：在 crontab 环境里面指定环境变量，如 PATH=/bin：/usr/bin：/usr/sbin。也可以通过“:”导入更多的环境变量。
- hour：运行 crontab 的小时，可设置成 0 ～ 23，单位是小时。

- minute：运行 crontab 的分钟，可设置成 0 ～ 59，单位是分钟。
- month：设置 crontab 运行的月份，可设置成 1 ～ 12，单位是月。
- monthday：一个月份中的哪一天，可设置成 1 ～ 31，单位是日。
- weekday：运行 crontab 的星期数，可设置成 0 ～ 7，单位是天。
- name：crontab 的注释，注释用于帮助管理员区分不同的 crontab。
- provider：默认值为系统自带的 crontab 程序。通常不需要指定此参数值，Puppet 会默认匹配系统自带的定时管理任务程序。
- user：将 crontab 加入某一个系统账号中，默认是加入执行守护进程的系统账户中。

2. cron 资源案例

通过 cron 资源设置每 5 分钟调用一次 ntpdate 调整系统时间，ntpdate 命令主要功能就是调整 UNIX/Linux 系统的时间和时区。编辑 /etc/puppet/manifests/cron.pp 文件，将以下内容追加到文件中。

```
cron {'ntpdate':
  ensure => true,
command =>'/usr/sbin/ntpdate 192.168.0.1',
    user =>'root',
  minute =>'*/5',
}
```

cron 资源的 command 属性指定调用的 ntpdate 命令和 IP 地址；user 属性指定系统执行 cron 的账户为 root；ensure 属性表示设置此 crontab，其实也可以忽略这一属性，因为其默认是启用的，当想关闭 crontab 时可以设置此属性为 false；minute 表示执行的时间，单位是分钟，与 crontab 的格式是一样的。设置完成后通过 puppet apply 命令来进行测试，过程和结果如下：

```
# puppet apply /etc/puppet/manifests/cron.pp
notice: /Stage[main]//Cron[ntpdate]/ ensure: created
notice: /Stage[main]//Cron[logrotate]/minute: minute changed '0' to '*/5'
notice: Finished catalog run in 0.02 seconds
```

通过 Puppet 输出，可以看到已经成功地创建了 crontab。可以通过 crontab 命令来确认是否执行成功。

```
# crontab -u root -l
# Puppet Name: ntpdate
*/5 * * * * /usr/sbin/ntpdate 192.168.0.1
```

7.2.9 notify 资源

notify 资源主要用于输出 Puppet 的辅助提示信息，在 Puppet 执行过程中通过这些辅助

信息了解执行的过程，它并不会改变任何操作状态。下面来看一下它的属性和案例。

1. notify 资源常用属性

以下为 notify 资源的常用属性。

```
notify{ ' 资源标题 ':
 name
 message
}
```

下面简单分析一下上面属性的作用。

- name：标识名。
- messages：输出描述信息。

2. notify 资源案例

这里以 notify 资源输出辅助信息为例。编辑 /etc/puppet/manifests/notify.pp 如下：

```
$sum = 8 + 64
notify{'sum':
  message => "sum:${sum}",
}
```

计算 8 加 64 的结果通过 notify 资源输出。执行 puppet apply /etc/puppet/manifests/notify.pp，结果如下。

```
# puppet apply /etc/puppet/manifests/notify.pp
notice: sum:72
notice: /Stage[main]//Notify[sum]/message: defined 'message' as 'sum:72'
notice: Finished catalog run in 0.02 seconds
```

Puppet 会输出 8 加 64 的累加和，同时可以在 message 属性中增加辅助提示信息，让输出更加人性化。

7.3 资源公有属性

Puppet Metaparameters 中文翻译为“元参数”，这里为了方便读者的理解，笔者又将它称为资源公有属性（下称资源公有属性），它包含的属性是在所有资源或类中可以通用的属性。Puppet 将资源完成任务与否的结果看为状态，正是这种特性让我们可以通过资源公有属性来建立资源与资源的关系。当建立了资源的关联关系后，前者资源状态未成功时，后者资源可以通过资源公有属性来确认其最终执行结果，这就是资源公有属性的用途。

目前 Puppet 提供了 10 多种资源公有属性，每种资源公有属性完成的任务是不一样的。下面首先介绍资源公有属性的应用场景。然后介绍 Puppet 常用的资源公有属性，它们分别是

before、require、stage、notify、subscribe 和 audit，其中 before 和 require 与 notify 和 subscribe 两对资源公有属性各为一组，前者表示先后顺序，后者描述了通知状态；stage 资源公有属性也是描述先后顺序的，不过它的范围更大些，可以调整类与类之间的顺序；audit 资源用于资源变更的审计工作。接着介绍资源公有属性通过符号定义的方式，通过符号定义公有属性从代码实现上也会更加的简洁。最后介绍通过资源公有属性来定义 chaining 链，即将所有的公有属性状态串联起来使用。关于资源公有属性的更多信息请参考官方网站 http://docs.puppetlabs.com/references/latest/metaparameter.html。

7.3.1 资源公有属性应用场景

通过一个实际的案例来看看什么情况下适合使用 Puppet 的资源公有属性。首先通过 Puppet 的 user 资源在机器上创建 test 账户，接着创建 test 账户的宿主目录，这时我们希望的结果是创建 test 账号成功后再创建它的宿主目录，否则如果 test 账号创建失败则不能创建它的宿主目录。为了介绍资源公有属性的使用方法，这里先模拟创建账号失败的场景，调用 user 资源时将 gid 属性设置为系统不存在的 gid 即可。编辑 /etc/puppet/manifests/user2.pp 文件，将以下内容追加到文件中。

```
user {'test':
  uid => '800',
  gid => '800',     # 指定系统不存在的 gid
  home => '/home/test',
}
file {'/home/test':
ensure => directory,
 group => '800',
 owner => '800',
}
```

/etc/puppet/manifests/user2.pp 文件的含义：首先通过 user 资源在系统创建 test 账号，并设置 test 账号的 uid/gid 分别为 800；设置 test 账号的宿主为 /home/test 目录，接着通过 file 资源创建 /home/test 目录，并设置 uid/gid 分别为 800。通过 puppet apply /etc/puppet/manifests/user2.pp 命令来测试它的最终结果，执行的过程和结果如下：

```
# puppet apply /etc/puppet/manifests/user2.pp
err: /User[test]/ensure: change from absent to present failed: Could not create user test: Execution of '/usr/sbin/useradd -u 800 -g 800 -d /home/test -s /bin/bash test' returned 6: useradd: Unknown group `800'.  # 未知的系统组 uid
notice: /Stage[main]//File[/home/test]/ensure: created   # 成功创建 test 账号宿主目录
notice: Finished catalog run in 0.03 seconds
```

根据 Puppet 的输出信息很容易看出，系统提示账号由于未知 800 的 gid 组，最终导致 test 账号创建失败，但是 test 账号的宿主目录却成功创建了，这是不符合我们需求的。我们希望的结果是 test 账号创建成功后，账号宿主目录才会创建，否则就不会创建账号的宿

主目录。那么如何达到我们的目的呢？这时就可以用上资源公有属性了。首先清理一下刚创建的 test 账号的宿主 /home/test 目录，然后修改 /etc/puppet/manifests/user2.pp 文件中的 Puppet 代码，增加 require 资源公有属性。这里先忽略 require 的作用，稍后会详细介绍。

```
# cat /etc/puppet/manifests/user2.pp
user {'test':
  uid => '800',
  gid => '800',
  home => '/home/test',
}
file {'/home/test':
 ensure => directory,
  group => '800',
  owner => '800',
require => User['test'],    # User 资源首字母大写
}
```

在 /etc/puppet/manifests/user2.pp 的 Puppet 代码中增加 require=> User['test']，它的含义是告诉 file 资源在执行前，确认标题为 test 的 user 资源是否为成功状态，如果是成功状态则继承执行，否则跳过本次执行。这里需要注意，调用 user 资源首字母“U”要大写。再次执行 puppet apply 命令，过程和结果如下：

```
# puppet apply /etc/puppet/manifests/user2.pp
err: /Stage[main]//User[test]/ensure: change from absent to present failed: Could not create user test: Execution of '/usr/sbin/useradd -u 800 -g 800 -d /home/test test' returned 6: useradd: Unknown group `800'.    # 异常，未知的系统组
notice: /Stage[main]//File[/home/test]: Dependency User[test] has failures: true
warning: /Stage[main]//File[/home/test]: Skipping because of failed dependencies    # 跳过执行
notice: Finished catalog run in 0.03 seconds
```

根据 Puppet 的输出信息，可以看到 test 账号由于 800 组不存在的原因仍然没有创建成功，创建 test 账号的宿主目录功能也跳过了，并提示错误信息（Skipping because of failed dependencies），这才是我们希望得到的结果。通过 require 资源公有属性实现了资源按顺序执行功能，也就是说当主调资源状态没有执行成功时，被调资源结果也不会成功；反之主调资源执行成功，被调资源才会成功，这就是资源公有属性的主要用途。

注意 在定义资源与资源间关联关系时，资源公有属性调用的资源首字母要大写，如以上案例中的 User ['test']，这里的首字母是必须大写的。

7.3.2 before 和 require 资源公有属性

笔者将 before 和 require 这两个资源公有属性分为了一组，因为它们的主要功能都是定

义资源与资源之间的先后顺序。

1. before 资源公有属性

Before 资源公有属性的作用是当前者资源成功执行后，再通知下一资源执行。以下为代码片段。

```
file {'/tmp/test1':
  ensure => present,
  content => 'Hi.',
  before => Notify['/tmp/test1 has already been synced.'],  # Notify 资源首字母大写
}
notify {'/tmp/test1 has already been synced.': }
```

当 file 资源成功将“Hi.”内容写入 /tmp/test1 文件中后，则调用 notify 资源将“/tmp/test1 has already been synced.”内容输出到终端上。否则当 file 资源执行失败（失败的原因有多种，如磁盘满了、inode 满了和磁盘故障等），最终 notify 资源则跳过执行。这就通过 before 资源公有属性实现了资源与资源之间的关联关系。

2. require 资源公有属性

Require 资源公有属性和 before 资源公有属性恰好相反，其主要功能是在本资源执行之前，需要确认其他资源是否已经被成功执行。以下为代码片段。

```
file { '/usr/local/scripts':
 ensure => directory
}
file { '/usr/local/scripts/myscript':
 source => 'puppet: //server/module/myscript',
 mode => '755',
 require => File["/usr/local/scripts"],    # File 资源首字母大写
}
```

第一个 file 资源标题为 /usr/local/scripts/myscript，ensure 属性为 directory，它的主要的功能是创建 /usr/local/scripts/myscript 目录结构；第二个 file 资源在执行前先要确认第一个 file 资源标题为 /usr/local/scripts 的目录是否已经被成功创建，成功创建后才会同步 myscript 脚本文件到 /usr/local/scripts/myscript 目录下。反之如果第一个资源创建失败，由于 require 资源公有属性的限制，第二个资源也就不会执行了，这样就保证 Puppet 同步配置文件时不会出现目录不存在的问题。

7.3.3 notify 和 subscire 资源公有属性

笔者将 notify 和 subscribe 这两个资源公有属性分为一组来介绍，因为它们的主要功能都是资源与资源之间状态的通知，notify 资源公有属性为主动通知，subscire 资源公有属性

为被动通知。

1. notify 资源公有属性

Notify 资源公有属性的主要作用是用来主动通知其他资源本资源的状态。以下为代码片段。

```
file { '/etc/ssh/sshd_config':
  ensure => file,
  source => 'puppet: ///modules/ssh/sshd_config',
  notify => Service['sshd'],     # Service 资源首字母大写
}
service { 'sshd':
  ensure => running,
  enable => true,
}
```

/etc/ssh/sshd_config 配置文件同步成功后会主动通知 servcie 资源重新加载配置文件。否则 file 资源执行失败，service 资源则跳过执行。这就保障了在没有加载最新配置的情况下，sshd 进程不会重新启动。

2. subscire 资源公有属性

Subscire 资源公有属性和 notify 资源恰好相反，它用于被动通知，当 subscribe 资源公有属性检测依赖资源变化时，会主动更新所在资源符状态。以下为代码片段。

```
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => 600,
  source => 'puppet: ///modules/ssh/sshd_config',
}
service { 'sshd':
  ensure => running,
  enable => true,
  subscribe => File['/etc/ssh/sshd_config'],     # File 资源首字母大写
}
```

当 service 资源检查 /etc/ssh/sshd_config 配置文件时发现变更，则加载变更后的配置，并重新启动 sshd 守护进程。否则检查 /etc/ssh/sshd_config 配置文件没有变更，则跳过 service 资源的执行。

7.3.4 资源公有属性的其他描述方式

Puppet 的资源公有属性中还可以通过“ ->”和“ ~>”两种特殊符号来描述资源与资源之间的关系。通过符号的描述方式会让代码看起来更加简洁。但不管是符号方式还是英文单词描述方式，其最终的结果都是一样的，即串联资源与资源的关系。两种符号描述方式

的含义如下。

- -> ：用于表示资源与资源之间的先后关系，等同于 before 和 require 两个资源公有属性。
- ~>：用于表示资源之间的通知，等同于 notify 和 subscire 两个资源公有属性。

下面通过资源公有属性符号的形式来介绍其他两种描述资源与资源之间关系的案例。

案例 1

通过 -> 符号来描述资源与资源之间的前后顺序关系。以下为代码片段。

```
file {'/tmp/test1':
  ensure => present,
  content => 'Hi.',
}
notify { 'after':
  message => '/tmp/test1 has already been synced.',
}
File['/tmp/test1'] -> Notify['after']     # File与Notify资源首字母大写，其中 -> 符号用来表示资源前后关系
```

首先定义 file 资源，作用是在 /tmp/test1 目录下创建文件，并向文件内追加“ Hi.”内容，然后通过 File['/tmp/test1'] -> Notify ['after'] 实现资源与资源之间的前后顺序关系。当 file 资源标题为 /tmp/test1 且执行成功后，执行 notify 资源标题为 after。这里注意，资源与资源建立先后顺序关系时首写字母要大写。

两个资源 File 和 Notify 首字母都要大写。

案例 2

通过 ~> 来通知资源与资源的关系。以下为代码片段：

```
file {'/tmp/test1':
  ensure => present,
  content => 'Hi.',
}
~>      # ~> 符号用来表示通知
notify {'after':
  message => '/tmp/test1 has already been synced.',
}
```

首先定义 file 资源，作用是在 /tmp/test1 目录下创建文件，并向文件内追加“ Hi.”内容，通过 ~> 通知 notify 资源标题为 after 的执行，显示成功同步的信息。否则 file 资源执行失败，则 notify 资源跳过执行。

7.3.5 定义 Chaining

Chaining 中文译为“链”（下称“链”），可以将资源与资源之间的状态串联起来定义，

资源与资源之间的依赖关系称为链，如图 7-4 所示。下面通过安装 openssh-server 软件的方式来介绍定义链的两种方式。

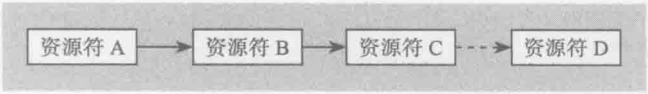


图 7-4 Puppet 的链

1. 资源公有属性方式

首先通过 package 资源来安装 openssh-server 软件包。openssh-server 软件包安装成功后再同步 sshd_config 的配置文件。成功同步 sshd_config 配置文件后，最后重启 sshd 守护进程。

安装 openssh-server 软件包。

```
package { 'openssh-server':
  ensure => present,
  before => File['/etc/ssh/sshd_config'],    # File 资源首字母大写
}
```

如果 openssh-server 安装成功，则通过 file 资源同步 sshd_config 配置文件。

```
file { '/etc/ssh/sshd_config':
  ensure => file,
   mode   => '600',
  source => '/root/examples/sshd_config',
  notify => Service ['sshd'],    # Service 资源首字母大写
}
```

成功同步 sshd_config 配置文件后，则通知 service 资源重新加载 sshd 守护进程。

```
service { 'sshd':
  ensure  => running,
  enable  => true,
}
```

2. 资源公有属性的符号方式

这里和上边“资源公有属性方式”中的案例一致，只不过通过 -> 和 ~> 两种符号来描述资源与资源之间的关系。

```
package { 'openssh-server':
  ensure => present,
}
file { '/etc/ssh/sshd_config':
  ensure => file,
    mode => '600',
```

```
  source => 'puppet: ///files/ssh/sshd_config',
}
service { 'sshd':
  ensure => running,
  enable => true,
}
# Package File Service 首字母大写
Package['openssh-server'] -> File['/etc/ssh/sshd_config '] ~> Service['sshd']
```

当 Package['openssh-server'] 执行成功后，通过 -> 执行 ['/etc/ssh/sshd_config'] 资源，当 File['/etc/ssh/sshd_config'] 资源执行成功后，通过 ~> 通知 Service['sshd'] 重新加载守护进程。

7.3.6 stage 资源公有属性与 stage 资源

stage 资源公有属性通常与 stage 资源共同使用，stage 资源又被看成运行阶段资源，比 before 和 require 资源公有属性范围更大，可以调整类与类之间的执行顺序。使用 stage 运行阶段资源分为以下两个部分：

- 定义 stage 资源类型。
- 通过 stage 资源公有属性衔接类名。

以下为 stage 运行阶段代码片段。

1）定义 stage 资源类型。

```
stage {"pre": before => Stage["main"]}
stage {"last": require=>Stage["main"]}
```

2）衔接类之间的关系。

```
class db_install{....}                  # 安装数据库代码
class web_install{....}                 # 安装 web 代码
class {"db_install": stage => "last"}
class{"web_install": stage => "pre"}
```

它的执行顺序为：先是 pre，然后是 main，最后是 last。我们经常在 Puppet 输出中看到 main 的身影，它是 Puppet 默认运行阶段。

以下为 stdlib 中的 stage 应用案例，其中 -> 表示通知类之间的先后顺序。

```
stage { 'setup':  before => Stage['main'] }
stage { 'runtime': require => Stage['main'] }
-> stage { 'setup_infra': }
-> stage { 'deploy_infra': }
-> stage { 'setup_app': }
-> stage { 'deploy_app': }
-> stage { 'deploy': }
```

7.3.7 audit 审计

audit 资源公有属性主要用于资源属性的审计，当某资源变化状态变化时，它可以将变化的内容追加到系统日志中。具体使用方式如下：

```
file{'/etc/passwd':
  audit => [ owner,mode ],
}
```

Puppet 执行过程中会记录和监视 /etc/passwd 文件的 owner 和 mode 两个权限，如果更改 /etc/passwd 文件的这两个权限，更改的记录就会被记录到系统日志中。

```
notice: /Stage[main]//Node[puppet_agent]/File[/etc/passwd]/mode: audit
change: previously recorded value 644 has been changed to 666
```

同时我们也可以监控系统日志的变化，并将变化的信息通过 Puppet 邮件功能推送给管理员。

7.4 默认资源

默认资源可以为资源初始化属性和值，通常默认资源声明在 site.pp 文件的首部，代码如下：

```
#site.pp
Exec {
  path => '/usr/bin:/bin:/usr/sbin:/sbin',
}
```

声明默认资源注意事项如下：

- 声明默认资源时首字母需要大写，如 exec 声明默认资源 Exec、package 声明默认资源 Package 等。
- 如果声明资源有一个名称空间资源“::”，它的每个环节都需要首字母大写，如 Concat::Fragment。

看一下 exec 和 package 两个资源声明默认资源的方法。

Exec 默认资源的声明方法如下：

```
Exec { path => '/usr/bin:/bin:/usr/sbin:/sbin' }     # Exec 默认资源首字母大写
exec { 'echo this works': }
```

通过 Exec 默认资源声明 path 属性的环境变量值，在后续声明 exec 资源时可以直接调用系统命令而不用担心环境变量的问题。

Package 默认资源的声明方法如下：

```
Package { provider => 'rpm'}     # Package 默认资源首字母大写
```

```
package { 'nginx': }
```

在默认资源中声明 provider 属性，指定包的安装方式为 rpm，后续 package 资源中 provider 属性均为 rpm。

7.5 Puppet 虚拟资源

Puppet 中的 Virtual Resource（中文译为“虚拟资源”，下文统称“虚拟资源”）主要用来解决资源重定义的问题。首先通过一个案例来了解虚拟资源应用场景；然后了解如何声明虚拟资源并通过虚拟资源来解决 Puppet 应用中的实际问题。

7.5.1 虚拟资源应用场景

通过一个案例来介绍 Puppet 虚拟资源的应用场景。在一个基础 base::acounts 类内集中通过 user 资源创建系统账号，并通过 webserver 节点来引用这个基础类。希望的结果是在 webserver 节点上只创建 testing 账号。但实际结果是什么样的呢？

根据之前所学习的知识，首先定义类名为 base::account，此类的逻辑主要负责创建账号，在 base::account 类中通过资源来分别创建 development 开发账号和 testing 测试账号，然后调用测试 webserver 节点来加载这个基础类。以下为代码片段。

```
# /etc/puppet/manifests/site.pp
class base::accounts{
  user {'development':     # 创建 developemnet 账号
    ensure => present,
    home => '/data/home/development',
    shell => '/bin/bash',
  }
  user {'testing':     # 创建 testing 账号
    ensure => present,
    home => '/data/home/testing',
    shell => '/bin/bash',
  }
}
node webserver{
   include base::accounts
}
```

我们希望得到的结果是在 webserver 的 Agent 上创建 testing 系统账号，但是实际执行 Master 的配置结果如下：

```
info: Caching catalog for test_webserver
info: Applying configuration version '1379987749'
notice: /Stage[main]/Base::Accounts/User[development]/ensure: created   #
development 账号被创建
```

```
notice: /Stage[main]/Base::Accounts/User[testing]/ensure: created  # testing账号被创建
notice: Finished catalog run in 0.05 seconds
```

在 webserver 的服务器上分别创建了 testing 和 development 两个账号，所以创建两个账号的结果并不符合我们需求的。这时就可以使用 Puppet 资源提供的“虚拟资源”功能，来解决这个问题。

7.5.2 虚拟资源

虚拟资源与普通资源的区别是，虚拟资源定义后要先实例化再使用，而普通资源定义后可以直接使用。定义虚拟资源的方法是在资源前追加 @，如 @user，这时的 user 资源就是一个虚拟资源。在代码文件中将资源转为虚拟资源后，Puppet 在执行的时候并不会调用它，如果想执行，需要通过 realize 函数或者“<| |>”（又称飞船方法）来实例化一个虚拟资源。下面通过 3 个案例来具体介绍虚拟资源的使用。具体内容如下。

- 案例 1：改造 7.5.1 节中的案例，通过虚拟资源方式来创建用户。
- 案例 2：利用虚拟资源解决资源重定义的情况。
- 案例 3：虚拟资源的其他描述方式。

案例 1

这里来改造 7.5.1 节中的案例。我们希望的是在 webserver 节点上只创建 testing 账号，在 user 资源前追加“@”。详细代码如下：

```
# /etc/puppet/manifests/site.pp
class base::accounts{
   @user {'development':     # 创建development账号
      ensure => present,
        home => '/data/home/development',
       shell => '/bin/bash',
   }
   @user {'testing':     # 创建testing账号
      ensure => present,
        home => '/data/home/testing',
       shell => '/bin/bash',
   }
}
node webserver{
   include base::accounts
   realize(User['testing'])  # 通过realize函数实例化User资源，注意User首字母需要大写。
}
```

在 webserver 的 Agent 上执行结果如下：

```
info: Caching catalog for test_webserver
info: Applying configuration version '1379987749'
notice: /Stage[main]/Base::Accounts/User[testing]/ensure: created
notice: Finished catalog run in 0.05 seconds
```

通过虚拟资源满足了我们之前的需求，即在 webserver 节点上创建 testing 账号。通过最终的结果可以看出，当资源声明为虚拟资源时，Puppet 在执行过程中并不会直接调用，只有通过 realize 实例化后的资源才会被调用。

案例 2

下面再来看一个资源标题重定义的案例。

为 webserver 节点安装 Nginx 软件包，但 webserver 节点中的 Agent 包含了 online（在线服务器）和 offline（离线服务器），通过 webserver 节点来声明它们，在 site.pp 文件中分别创建两个类——app::online 类（在线服务器）和 app::offline 类（离线服务器）。代码如下：

```
class app::offline{
  include offline     # 加载离线服务器相关信息
  package {'nginx-server': ensure => installed }     # 安装 Nginx
}
class app::online{
  include online     # 加载在线服务器相关信息
  package { 'nginx-server': ensure => installed }     # 安装 Nginx
}
```

如果分别加载 app::online 和 app::offline 这两个类是没有问题的，但这里将两个类放到一个 webserver node 节点中使用。代码如下：

```
node webserver{
  include app::offline
  include app::online
}
```

在 webserver 的 Agent 访问 Master 的配置信息时，就会报以下的错误信息：

```
err: Could not retrieve catalog from remote server: Error 400 on SERVER:
Duplicate declaration: Package[nginx-server] is already declared in file
```

上述的执行结果是一条错误提示，错误类型为资源重定义。之所以会出现这类错误，是因为用户试图用同样的资源标题 nginx-server 来安装软件，这样是 Puppet 不允许的。除非更换创建用户资源标题的名字，但是其他类在加载的时候就会出现问题，怎么办？这时就可以使用 Puppet 的虚拟资源功能来完成这个任务。以下为通过虚拟资源改造后的代码。

首先定义一个虚拟包 admin::virtual-packages 类，将 package 转为虚拟资源。

```
class admin::virtual-packages{
  @package { "nginx-server": ensure => installed }
}
```

然后分别在另两个类中通过系统函数 realize 来调用 package 类的标题 nginx-server。

```
class app::offline{
 # 通过 realize 实例化资源，注意实例后的 Package 资源首字母大写
 realize(Package['nginx-server'])
```

```
}
class app::online{
 realize(Package['nginx-server'])     # 同上
}
```

通过虚拟资源来实例化资源的方法解决了资源重定义的冲突。再次执行 Agent 会发现，刚刚的错误信息消失了。

案例 3

实例化一个虚拟资源除了用系统提供的 realize 函数外，还可以用“<| |>”。

```
class admin::virtual-packages{
  @package { "nginx-server": ensure => installed }     # 声明虚拟资源
}
class app::offline{
  Package<| title == "nginx-server"|>     # 实例化资源，结果同 realize(Package['nginx-server']) 一致
}
class app::online{
  Package<| title == "nginx-server"|>     # 同上
}
```

在 web_server 节点中加入 admin::virtual-packages，重新执行 Agent 也是正确的。

```
node web_server{
  include admin::virtual-packages
  include app::offline
  include app::online
}
```

7.6 Puppet 资源的导出

Puppet 支持资源导出的功能，此功能可以将 Agent 编译后的目录在其他任意 Agent 上收集与创建。常见的案例如 A/B 两台 Agent，Puppet 可以将两台 Agent 的 SSH 公钥导出，并交换这些信息，分别在 A/B 上创建互相的公钥。这样的配置可以提高安全性能，并消除第一次使用 SSH 登录的 unknown host 警告。目前 Puppet 资源导出后可以存放在 3 种存储中，它们分别是 MySQL、SQLite3 和 PuppetDB，其中 MySQL 配置最简单。本节主要以 MySQL 作为存储来介绍 Puppet 资源的导出。

7.6.1 环境的配置

1. 环境安装

首先安装资源导出的环境。我们以 CentOS 发行版本为例，在 Master 安装 MySQL 的环境。

```
# yum install mysql
# yum install mysql-devel
# yum mysql-server
# yum ruby-mysql
```

Puppet 中的资源导出与存储利用了 Ruby on Rails 框架，它将 Puppet 资源模块化后存储到一个支持关系型数据库的 Active Record 中，所以还要安装 Ruby on Rails 与 activeRecord 库。接着通过 gem 来安装它们。

1）Ruby on Rails 安装（Puppet 0.2.48 与 Puppet 0.25.* 需要安装 rails 的 2.2.2 以上版本）。

```
# gem install rails -v 2.3.5 -no-ri -no-rdoc
```

2）activerecord 安装。

```
# gem install activerecord -v 2.3.5 --no-ri --no-rdoc
```

2. 环境配置

1）MySQL 的配置。通过 Puppet 导出的所有的资源会存储到 MySQL 中，所以需要创建存储资源所用的 MySQL 库，并创建 Puppet 连接 MySQL 时所用的账户。

```
# mysql
mysql> create databases puppet    # 创建
# 创建访问账户与账户的密码
mysql> grant all privileges on puppet.* to puppet@localhost identified by
'puppet'
# 刷新配置
mysql> flush privileges;
```

2）让 Puppet 的配置支持资源导出与存储。修改 Master 的 puppet.conf 主配置文件，追加以下内容：

```
[master]
storeconfigs = true     # 开启存储配置
dbadapter = mysql     # 指定存储介质
dbname = puppet      # 指定访问 MySQL 数据库名
dbuser = puppet      # 指定访问 MySQL 的账户名
dbpassword = puppet       # 指定访问 MySQL 的密码
dbserver = localhost     # 指定 MySQL 的地址
dbsocket = /var/lib/mysql/mysql.sock    # 指定 MySQL 的 mysql.sock
```

Puppet 会根据此配置信息对导出后的资源进行存储。如果我们改变了 MySQL 数据库地址、账号或密码，需要同时变更 puppet.conf 配置文件中的配置。

7.6.2 资源导出案例

在一些网站或企业中新的系统上线，其他系统的 known 文件就会过时了，从而导致

SSH 登录时报 unknown host 警告。对于这个问题可以通过 Puppet 资源导出来解决。在新系统上线后，将其公钥追加到新的系统中，如图 7-5 所示。在 puppet2.example.com 上导入 pupet1.example.com 的 rsakey。

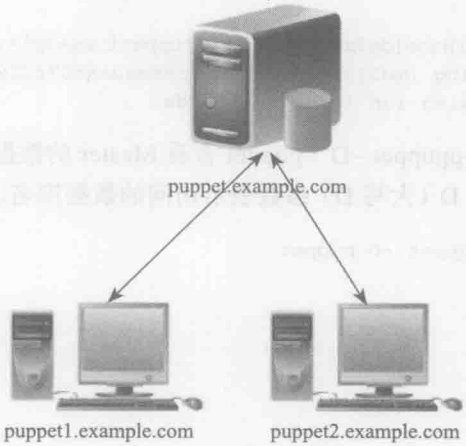


图 7-5 Puppet 资源导出

1. Master 配置

编辑 site.pp 文件，并追加以下内容：

```
node default{
  @@sshkey{"${fqdn}_rsa":     # 资源导出使用 "@@"
     host_aliases => ["$fqdn","$hostname","$ipaddress"],
     type => rsa,
     key => $sshrsakey,     # Agent 的 rsakey
   }
Sshkey <<| |>> {ensure => present}     # Sshkey 首字母大写，将导出的数据在机器上创建它
}
```

如以上代码所示，通过 sshkey 资源导出各机器的配置信息到存储中，导出资源要使用@@。标题中包含了 dsa 或 rsa，为了防止它们冲突，我们使用了 Facter 中的变量值。


注意 资源收集时指定的额外参数（Sshkey <<| |>> {ensure => present}）是在 Puppet 2.6.x 以后新增的功能，Puppet 0.25.* 不支持此功能。

2. Agent 配置

在 puppet1.example.com 上执行 puppet agent --server puppet.example.com --test 进行 Agent 配置。首次执行 Master 会根据 puppet.conf 创建相应的表，并将资源导出后的信息导入数

据库中。

```
# puppet agent --server puppet.example.com --test
info: Caching catalog for puppet1.example.com
info: Applying configuration version '1400907448'
# 创建 rsakey
notice: /Stage[main]//Node[default]/Sshkey[puppet1.example.com_rsa]/ensure: created
info: FileBucket adding {md5}313bdc7eb2d73cd6ea0a9507f22913f8
notice: Finished catalog run in 0.10 seconds
```

通过 mysql –upuppet –ppuppet –D puppet 查看 Master 的数据库，其中 u 参数表示登录名，p 参数表示登录密码，D（大写 D）参数表示访问的数据库名。

```
# mysql -upuppet -ppuppet -D puppet
mysql> show tables;
+------------------+
| Tables_in_puppet |
+------------------+
| fact_names       |
| fact_values      |
| hosts            |
| inventory_facts  |
| inventory_nodes  |
| param_names      |
| param_values     |
| puppet_tags      |
| resource_tags    |
| resources        |
| source_files     |
+------------------+
11 rows in set (0.00 sec)
```

如果管理超过 100 台的 Agent，建议为 reousrces 表添加索引，以提升查询的速度。添加索引的方式如下：

```
mysql> create index exported_restype_title on resources(exported,restype,title(50));
```

最后在 puppet2.example.com 上执行 puppet agent --server puppet.example.com --test。

```
# puppet agent --server puppet.example.com --test
info: Caching catalog for puppet1.example.com
info: Applying configuration version '1400907448'
# 在创建 puppet2.example.com_rsa key 基础上，在本机也创建了 puppet1.example.com_rsa 的 key
notice: /Stage[main]//Node[default]/Sshkey[puppet1.example.com_rsa]/ensure: created
notice: /Stage[main]//Node[default]/Sshkey[puppet2.example.com_dsa]/ensure: created
info: FileBucket adding {md5}313bdc7eb2d73cd6ea0a9507f22913f8
notice: Finished catalog run in 0.10 seconds
```

可以看到 puppet1.example.com 的 rsa 密钥同时在 puppet2.example.com 中也生成了一份。

7.6.3 过期资源清理

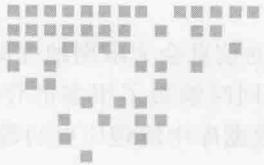
使用 Puppet 存储的潜在问题是，从 Agent 收集到的信息会无限期地存储在 MySQL 数据库中，久而久之就会影响 MySQL 数据库的性能，同时浪费了很多的存储空间。还好 Example.com 的操作成员为我们提供了工具，可以从数据库中清理历史的数据。工具下载地址如下：

```
# wget https://github.com/puppetlabs/puppet/blob/2.6.4/ext/puppetstoredconfigclean.rb
```

从数据库中清理一个节点历史数据，只需要将它们的主机名传给脚本就可以。使用方法如下：

```
# ruby puppetstoredconfigclean.rb mail{1,2,3}dev
```

运行配置清理脚本后，这些节点导出资源将不会再被任何 Puppet 清单文件所收藏。



Chapter 8 第 8 章

Puppet ERB 模板详解

现代人对工作的划分越来越细，而被划分出的小单元的功能越来越单一。人们之所以这样做，无非是想提高工作效率。以汽车的生产为例，汽车从开始制造到产出会经过一个流水线，每个流水线只负责自己相对独立的任务，每个人完成的部分连接到一起最后生产出一辆汽车，正是流水线确保了汽车的质量和生产效率。互联网时代也引入了这样的概念，网页编程中的模板技术就是如此。模板技术有很多种，有的模板技术可以实现动态静态分离，提高用户的访问速度，改善用户体验，降低服务器的压力，从而节约很多成本；有的模板技术可以提供一个网页模板，其他网页都基于这模板网页开发，这样既降低了开发成本，也节省了程序员的时间，提高了工作效率。同样 Puppet 也引进了模板技术，即 ERB 模板技术，而且拥有独立的 ERB 模板语言。Puppet 本身并没有模板概念，ERB 模板是 Ruby standard library 的一部分。

下面首先介绍 ERB 模板应用场景，让读者了解应该如何使用它；然后介绍 ERB 的语言；最后介绍 ERB 模板的案例，通过 ERB 模板配置 Apache 虚拟主机。

8.1 ERB 模板应用场景

Puppet 配置管理工具支持模板技术，但它并不自带模板，而是使用了 Embedded Ruby（简称 ERB）来支持模板技术。Puppet 引入 ERB 模板技术的优势是可以让我们更加方便地管理差异化的服务器配置信息，那么 ERB 模板技术如何差异化管理服务器呢？举个例子，我们通常将系统架构分为 3 层，分别是接入层、逻辑层和存储层。以接入层为例，我们要通过 Puppet 在接入层的 100 台服务器上安装 Apache 软件，按照我们之前掌握的知识，只

能通过 Puppet 在这些服务器上安装 Apache 软件，并没有提及如何差异化管理它们的配置文件信息。但是在日常运维工作中确实有很多这样的需求，如管理不同机器配置文件中的 IP、Apache 的连接数和虚拟主机等，它们各自有各自相对独立的差异化配置，如果要管理这些差异配置的话，显然之前掌握的知识不能满足需求，这时就可以通过 ERB 模板和模板语言来管理这些差异化服务器配置，这就是 ERB 模板的主要应用场景。

8.2 ERB 语言

本节主要结合 Puppet 配置管理功能来介绍 ERB 语言。首先来初识 ERB 模板，对 ERB 模板目录结构和使用先做一个基本的了解，然后再分别了解一下 ERB 语言中的变量、条件语句、循环语句和函数。关于 ERB 语言，更多信息可以参考官方网站 http://ruby-doc.org/stdlib-2.1.0/libdoc/erb/rdoc/ERB.html。

8.2.1 初识 ERB 模板

在 Puppet 配置管理系统中 ERB 模板的体现就是一个文本文件，它以 .erb 作为扩展名来标示它的用途。如从 Master 同步 resolve.conf 域名解析的配置文件到 Agent 上，代码如下所示：

```
file { "/etc/resolve.conf":
 content => template('resolve/resolve.erb'),
}
```

file 资源的标题指定了同步文件（/etc/resolve.conf）的目标路径，content 属性调用 template 函数指定了（resolve/resolve.erb）模板文件在 Master 的源路径（这里模板文件的路径可以使用绝对路径也可以使用相对路径，相对路径通常用于 C/S 架构，绝对路径通常用于单机，如 puppet apply 方式），执行 file 资源后 /etc/resolve.conf 配置文件就会根据 resolve/resolve.erb 模板文件中的内容进行同步。还可以在 resolve.erb 模板中追加条件语句，让某些机器在符合某些条件后才会生成最终文件。

```
<% if @in_var %>
nameserver 172.16.1.27
<% end %>
```

从 Master 同步文件到 Agent，我们也曾介绍过 file 资源中的 source 属性，与 template 函数调用模板形式，两者相比都可以实现同步文件的功能，但是过程和结果却不一样，file 资源的 source 属性同步文件通过 Puppet 的文件协议，将文件由源路径同步到目的路径，但是它并不能更改文件中的内容，而 template 函数同步文件的同时会参考模板文件内容，并融入了编程语言，从而可以实现根据需求来定制同步文件与内容，这样就更加灵活了，也

为差异化配置提供了更多的发挥空间。

ERB 语言常见语法的介绍如表 8-1 所示。笔者会在后续小节中详细介绍它的使用方法和案例。

表 8-1 ERB 语言常见标记

标 记	说 明	标 记	说 明
<%= Ruby expression %>	Ruby 表达式的值，如变量名	<%% or %% >	<%% 等价于 <%，%>%> 等价于 %>
<% Ruby code %>	Ruby 代码，如条件、循环等	<%-	等同于 <%，开始标签
<%# comment %>	注释	-%>	等同于 %>，闭合标签

8.2.2 变量

了解了 Puppet 的 ERB 模板后，再来了解一下它在 ERB 模板中的变量。ERB 语言支持变量，变量即可以改变的量。在 ERB 模板语言中变量以“<%=”作为开始，以“%>”作为结束，中间可以写变量的名字，如“<%= ip %>”中的 ip 就是一个变量。

我们来看一下 Puppet 是如何将变量传入 ERB 模板的。以修改 /etc/resole.conf 文件为例，通过 Puppet 代码将变量传入模板，并最终修改 /etc/resole.conf 文件。以下为 Puppet 代码文件和 ERB 模板文件，首先来看 Puppet 代码文件，编辑 /etc/puppet/modules/resole/manifests/resole.pp 文件，将以下内容追加到文件中。

```
$ip_1 = "192.168.1.7"
$ip_2 = "192.168.1.67"
file { "/etc/resolve.conf":
  content => template('resolve/resolve.erb'),
}
```

在 resole.pp 文件中声明变量 $ip_1 和 $ip_2，并分别赋值，将 192.168.1.7 赋值给 $ip_1 变量，将 192.168.1.67 赋值给 $ip_2 变量；然后调用 file 资源，file 资源标题 (/etc/resolve.conf) 为要修改的目标文件，资源中的 content 属性指定 ERB 模板的位置，这里可以写相对路径也可以写绝对路径，如果写相对路径 Puppet 会到 /etc/puppet/modules/（模块名）目录 / 下寻找 *.erb 文件，如果写绝对路径 Puppet 就会直接加载绝对路径的文件名。这里以相对路径的形式来编辑 erb 文件。接着看 ERB 模板文件的内容，编辑 /etc/puppet/modules/resole/templates/resolve.erb，将以下内容追加到文件中。

```
nameserver <%= ip_1 %>
nameserver <%= ip_2 %>
```

resolve.erb 文件中的 <%= ip_1%> 和 <%= ip_2 %> 表示变量名，它们对应的变量值是 resolve.pp 文件中的 $ip_1 和 $ip_2 的值，即变量 <%= ip_1%> 的值为 192.168.1.7，变量 <%=ip_2%> 的值为 192.168.1.67。通过 puppet apply /etc/puppet/modules/resole/manifests/resole.pp 方式来执行文件，结果如下：

```
#puppet apply /etc/puppet/modules/resole/manifests/resole.pp
notice: /Stage[main]//File[/etc/reslolve.conf]/content: content changed '{md5}7b
0140d14347aacc94c89a76be6f9cca' to '{md5}a3ecce9717f32bf1b3001195a9c306ae'
notice: Finished catalog run in 0.02 seconds
```

成功执行后，可通过 cat 命令来确认一下 Puppet 的执行结果，将 /etc/resolve 域名解析文件按照我们的 resolve.erb 模板文件进行解析，生成的最终结果如下：

```
# cat /etc/resolve.conf
nameserver 192.168.1.7
nameserver 192.168.1.67
```

8.2.3 if…elsif…else 条件语句

下面来介绍一下 ERB 模板语言的条件语句 if…elsif…else 的语法和案例。

1. if…elsif…else 语法

if…elsif…else 条件语句需要放入以 <% 作为开始，以 %> 作为结束的符号内。另外需要注意 <% if 条件表达式 %> 最后要以 <% end %> 作为结束。以下为 if…elsif…else 的语法。

```
<% if 条件表达式 1 %>
   执行语句 1
<% elsif 条件表达式 2 %>
   执行语句 2
<% else %>
   执行语句 3
<% end %>
```

以上为 if…elsif…else 条件判断语句的语法，大致的判断流程如下：条件语句首先判断条件表达式 1 是否成立，如果成立则加载执行语句 1；如果不成立则判断条件表达式 2 是否成立，如果成立则加载执行语句 2；如果以上两个条件都不成立，最后加载执行语句 3。

2. if…elsif…else 案例

以 postfix 的配置文件模板为例来介绍条件语句的使用场景。在安装 postfix 时根据操作系统发行版本，增加不同特性的配置内容。首先来看一下 postfix 的 Puppet 代码，编辑 /etc/puppet/modules/postfix/manifests/postfix.pp 文件，内容如下：

```
class postfix::install{
 package{ ["postfix","mailx"]:
  ensure => present,
 }
}
class postfix::config{
```

```
file{"/etc/postfix/master.cf":
    ensure => prensent,
   content => template("postfix/master.erb"),
  }
}
```

postfix::install 类的作用为安装 postfix 邮件服务器程序，postfix::config 类的作用为同步 master.cf 到 Agent 服务器，并根据 Agent 的操作系统发行版本生成相应的配置文件。来看一下 master.erb 模板文件内容，编辑 /etc/puppet/modules/postfix/templates/main.cf.erb 配置文件，内容如下：

```
<% if operatingsystem == "FreeBSD" %>
 # FreeBSD
 alias_maps = hash:/etc/mail/aliases
 alias_database = hash:/etc/mail/aliases
 daemon_directory = /usr/local/libexec/postfix
 command_directory = /usr/local/sbin
<% elsif operatingsystem == "RedHat" %>
 # RedHat
 alias_maps = hash:/etc/aliases
 alias_database = hash:/etc/aliases
 daemon_directory = /usr/libexec/postfix
 command_directory = /usr/sbin
<% else %>
 # Default
 alias_maps = hash:/etc/aliases
 alias_database = hash:/etc/aliases
 daemon_directory = /usr/lib/postfix
 command_directory = /usr/sbin
<% end %>
```

在 postfix.pp 文件中并没有声明 operatingsystem 变量，但在 main.cf.erb 模板文件中仍然能使用 operatingsystem 变量，这是因为它是 Puppet 的默认变量，Master 通过 facter 程序收集系统的信息，并保存到系统默认变量中。在 Puppet 中可以直接调用这些变量信息。以上模板程序的含义是当变量 operatingsystem 的值为 FreeBSD 时，加载 FreeBSD 系统的目录配置文件结构；当系统变量 operatingsystem 的值为 RedHat 系统时，则加载 RedHat 系统的目录配置文件结构；如果都未匹配到，则加载默认目录配置文件结构。为了看懂配置文件的变化，笔者通过“#”注释功能在不同操作系统发行版本中增加了相应的注释信息，执行 puppet apply /etc/puppet/modules/postfix/manifests/postfix.pp 代码。首先通过 facter 命令看一下我们操作系统发行版本，然后再来看一下 Puppet 根据操作系统发行版本最终生成的配置文件的内容。

通过 facter 命令来查看一下本机的 operatingsystem 的变量值（关于 facter 命令的详细内容会在第 9 章详细介绍，读者在这里只需要了解一下就可以）。以下为 facter 命令的结果。

```
# facter | grep operatingsystem
operatingsystem => SLES
```

operatingsystem => SLES 表示我们的系统发行版本为 SLES，即 Suse 发行版本。接着通过 cat 命令来查看一下通过 Puppet 模板生成的 /etc/postfix/master.cf 配置文件的内容。

```
# cat /etc/postfix/master.cf
#default
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
daemon_directory = /usr/lib/postfix
command_directory = /usr/sbin
```

可以看到，模板中最终的结果就是 #default 的内容，因为我们的发行版本并不是 Freebse 和 RedHat。

8.2.4 each 循环

ERB 模板语言中的循环语句为 each 循环。下面来介绍一下 each 循环的语法和案例。

1. each 循环语法

each 循环语句以 <% 作为开始，以 -%> 作为结束。格式如下：

```
<% [1,2,3,4,5,6,7,8,9].each do |val| -%>
        <%= val %>
<% end -%>
```

以上是 each 循环语句的语法，其中数组 [1，2，3，4，5，6，7，8，9] 作为数组的输入，val 作为遍历的值，它会依次遍历 1 ～ 9 的值并输出。这就是 each 循环的主用途——遍历数组。

2. each 循环案例

再来看一下 Puppet 是如何向 each 循环传递变量的。仍然以 resolve.conf 域名解析文件为例，通过 Puppet 数组向 ERB 模板文件传值变量来生成新的 resolve.conf 文件。首先来看 Puppet 代码文件，然后再来看 ERB 模板文件。编辑 /etc/puppet/resolve_array.pp 文件，将以下内容追加到文件中。

```
$arr_value=["192.168.1,1","192.168.1.2","192.168.1.3"]
file { "/etc/resolve.conf":
 content => template('resolve/resolve.erb'),
}
```

声明 $arr_value 数组并追加 IP 值为 ["192.168.1，1"，"192.168.1.2"，"192.168.1.3"]；调用 file 资源指定修为配置文件（/etc/resolve.conf）位置；调用 template 函数指定 ERB 模

板（'resolve/resolve.erb'）的位置。整个代码逻辑的最终的结果是将 IP 数组值传入 resolve.erb 文件中，并通过 resolve.erb 模板文件中的逻辑来继续处理。

继续来看 ERB 模板文件，编辑 /etc/puppet/modules/resolve/templates/resolve.erb 文件，将以下内容追加到文件中。

```
<% arr_value.each do |val| -%>
nameserver <%= val %>
<% end -%>
```

each 循环会根据 Puppet 的 arr_value 变量数组进行遍历，并将数组内容依次追加到 /etc/resolve.conf 文件中。执行 puppet apply resolve_array.pp 文件后，通过 cat 命令来确认看一下最终的结果。

```
# cat /etc/resolve.conf
nameserver 192.168.1.1
nameserver 192.168.1.2
nameserver 192.168.1.3
```

可以看到，已经将 $arr_value 数组变量的值通过 each 循环写入了 /etc/resolve.conf 配置文件中，通常当变量比较多而且它们又完成一类功能的时候可以使用这种循环方式。

8.2.5 函数

在 ERB 模板中还支持直接调用 Facter 的变量与 Puppet 的函数值，ERB 模板通过 scope::lookupvar 的方式来支持扩展更多的功能。首先来看一下如何调用 Facter 中的变量。如在 ERB 模板中调用 ipaddress 变量，变量中存放了机器的 IP 地址，可以通过 <%= scope::lookupvar('ipaddress') %> 的方式直接调用并应用到 ERB 模板中。

再来看如何在 ERB 模板中调用 Puppet 函数，在 ERB 模板调用 Puppet 函数必须以 scope.function_ 作为开始，后接 Puppet 的函数名（关于 Puppet 更多的函数请参考第 6 章函数介绍）。如我们调用 Puppet 的 warning 函数，此函数的作用是打印信息，它的调用方式为 scope.function_warning。仍然以 resolve.conf 文件为例介绍 ERB 模板中调用 Puppet 函数的方法，将 #this is resolve.conf 信息追加到生成的模板文件中。首先看一下 Puppet 代码，编辑 /etc/puppet/modules/resole/manifests/resolve，内容如下：

```
$arr_value=["192.168.1,1","192.168.1.2","192.168.1.3"]
file { "/etc/resolve.conf":
 content => template('resolve/resolve.erb'),
}
```

接着来看 ERB 模板文件，编辑 /etc/puppet/modules/resole/templates/resolve.erb 文件，将以下内容追加到文件中。

```
<%= scope.function_warning(["#this is resove.conf"]) %>
```

```
<% arr_value.each do |val| -%>
nameserver <%= val %>
<% end -%>
```

我们将 <%= scope.function_warning(["#this is resove.conf"]) %> 追加到模板文件中，它的含义是将“# this is resove.conf”信息一并编译到模板中。执行 resolve_array.pp 文件后，通过 cat 命令来查看一下结果。

```
# cat /etc/resolve.conf
# this is resove.conf
nameserver 192.168.1.1
nameserver 192.168.1.2
nameserver 192.168.1.3
```

可以看到“# this is resove.conf”内容已经被编辑追加到了生成的文件中。这里还可以通过它调用更多的 Puppet 函数来为我们解决一些日常工作中的实际问题。关于 Puppet 函数的更多信息请参考官方网站 http://docs.puppetlabs.com/guides/templating.html。

8.3 通过 ERB 模板配置 Apache 虚拟主机

本节以案例的形式来介绍如何通过 ERB 模板配置 Apache 虚拟主机。具体的实现步骤如下。

步骤 1 在 Master 上创建配置文件和目录。

```
# mkdir -p /etc/puppet/manifests/httpd/templates/
# mkdir -p /etc/puppet/manifests/httpd/files/
# mkdir -p /etc/puppet/manifests/httpd/tests/
# mkdir -p /etc/puppet/manifests/httpd/spec/
# touch /etc/puppet/manifests/httpd/manifests/init.pp
# touch /etc/puppet/manifests/httpd/templates/httpd.conf.vhost.reb
```

在第 5 章中曾经介绍过基础模块目录结果，这里就不赘述了。

步骤 2 在 Master 上编写配置 Apache 虚拟主机的代码。编辑 /etc/puppet/manifests/httpd/manifests/init.pp 文件，内容如下：

```
class apache::parameter{
$listenaddress = "${ipaddress}"
$server_admin = "admin@qq.com"
$server_name = "web.puppet.example.com"
$document_root = "/var/www/html/puppet"
}
```

apache::parameter 类存放了虚拟主机的变量值。下面来简单分析一下虚拟主机的变量值含义。

- listenaddress：使用了 Puppet 的默认 ${ipaddress} 变量，此值是 Agent 机器的 IP。

- server_admin：虚拟主机的管理员邮箱。
- server_name：虚拟主机的域名。
- document_root：虚拟主机的发布目录。

```
class httpd inherits apache::parameter{
package { "apache":
 ensure => installed,
}
file { "/usr/local/apache2/conf/httpd.conf":
 mode =>'777',
 content => template("puppet:///files/httpd/httpd.conf.vhost.reb"),
  notify => Service['httpd'],
}
service { "httpd":
   ensure => running,
hasrestart => true
 hasstatus => true
  }
}
```

httpd 类继承了 apache::parameter 类的相关变量，httpd 类中包含了 3 个资源。下面简单分析一下这 3 个资源的作用。

- package 资源：主要用于 Apache 的安装。
- file 资源：设置文件权限为可读、可写、可执行。将 apache::parameter 类中变量信息赋值到 http.conf.vhost.reb 模板变量中，并同步到指定服务器的 /usr/local/apache2/conf/httpd、conf。
- service 资源：主要用于同步 http.conf 配置文件后重启 Apache。

Puppet 的代码文件都准备好了，再来看一下 Apache 的 /etc/puppet/manifests/httpd/templates /httpd.conf.vhost.reb 模板配置文件。由于内容比较多我们只截取了虚拟主机配置部分。

```
<VirtualHost <%= listenaddress %>:80>
   ServerAdmin <%= server_admin %>
   DocumentRoot <%= document_root %>
   ServerName <%= server_name %>
   ErrorLog logs/demo.neoease.com-error.log
   CustomLog logs/demo.neoease.com-access.log common
</VirtualHost>
```

修改好 httpd.conf.vhost.erb 文件后，就可以通过 Ruby 提供的 ERB 工具来确认 httpd.conf.vhost.erb 文件的语法是否正确了。

```
# erb -P -x -T '-' httpd.conf.vhost.reb | ruby -c
Syntax OK
```

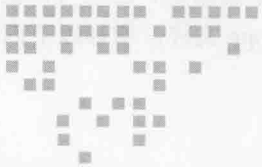
Syntax OK 表示 httpd.conf.vhost.reb 模板文件没有语法错误，已经可以使用了。

步骤 3 在 Master 上修改 /etc/puppet/manifests/site.pp 文件，增加以下内容：

```
node web.puppet.example.com {
 include httpd
}
```

当 Agent 为 web.puppet.example.com 并访问 Master 时，会自动加载 httpd 基础模块。

步骤 4 在 Agent 通过 puppet agent --server=puppet.example.com --test 来同步 Master 的配置时，Master 会优先安装 Apache，然后同步 httpd.conf 配置文件，根据 Apache 模板同步 Apache 的虚拟主机配置，最后重启 Agent 的 Apache 守护进程。



Chapter 9 第 9 章

走进 Facter

笔者曾在第 6 章中介绍过 Facter，但只介绍了它的常用变量而已。我们也曾在很多章中看到 Facter 的身影，它是 Puppet 配置管理服务器中的重要工具成员之一。本章将会详细介绍 Facter 这款强大的工具。为什么我们通过 Puppet 配置管理工具来管理服务器时，操作系统对我们来说是透明的？为什么 Puppet 配置管理工具可以收集并读取到 Agent 的机器信息？Puppet 如何通过收集到的 Agent 信息来区别化管理？这一系列的问题在本章都会找到相应的答案。通过对本章的学习，读者还会了解到 Facter 不仅可以收集 Agent 上的信息，还可以通过其他的编程语言来协助 Facter 收集更多的扩展信息。本章首先介绍 Facter 的作用、版本分支和使用方式等，让大家来认识它；然后采用分组的方式介绍 Facter 中的变量，让读者更方便记住它们；最后介绍扩展 Facter 常见的 3 种方式。

9.1 Facter 简介

Facter 是一款扩展性强且功能强大的跨平台的系统性能分析收集工具，由 Ruby 语言编写。它可以收集机器的信息，并将收集到的信息作为变量传给 Puppet 使用（在 Puppet 中这些变量均为 top 域变量，可以在 Puppet 语言中的任何位置调用它们。另外在 Puppet 中又将这些收集到的变量称为 Fact）。Facter 中收集的变量中包含系统的主机名、IP 地址与硬件地址、操作系统发行版本、内核版本、内存空间和磁盘空间等信息，如果 Facter 收集的变量中不能满足我们的需求，还可以通过 Ruby 语言来扩展它的变量，或者通过导入环境变量的方式来追加更多的信息，这就是 Facter。关于 Facter 更多信息可以参考 http://docs.puppetlabs.com/facter/。

9.1.1 Facter 版本

截至本书出版前，Puppet 官网为我们提供 4 种 Facter 的版本，分别是 1.5*、1.6*、1.7* 和 2.0.* 版本。下面来简要介绍一下这 4 种版本。

- Facter 1.5.* 版本：为 Facter 早起版本，它存在大量的 bug，部分的 bug 甚至会导致服务器产生僵尸进程，最终导致机器服务的不可用，所以并不推荐使用。
- Facter 1.6.* 版本：目前提供了 86 个变量，此版本支持 Puppet 3.* 与 Puppet 2.7.* 版本。它与 Facter 1.5.* 版本比较，修复了大量的 bug，同时增加了一些系统变量。关于 Facter 1.6.* 小版本变化，更多信息请参考官方网 http://docs.puppetlabs.com/facter/1.6/release_notes.html。
- Facter 1.7.* 版本：目前提供 94 个变量，此版本支持 Puppet 3.* 与 Puppet 2.7.* 版本。Facter 1.7.* 版本中修复了 1.6.* 版本中的很多 bug，与 Facter1.6.* 版本相比较它还增加了一些常用的系统变量。更令人振奋的是它还增加了 External Facts 的功能（读者可以理解为“扩展变量”），它可以通过其他的编程语言或数据格式，如 Python、Shell、Ruby 和 json、yaml 结构化文件等方式，并按照规定的格式导入 Facter 中来扩展它本身的变量。这一功能也让一些熟悉各种编程语言的运维工程师们能更方便灵活地扩展 Facter 的变量。关于 Facter1.7.* 版本变化的更多信息，请参考官方网站 http://docs.puppetlabs.com/facter/1.7/release_notes.html。
- Facter 2.0.* 版本：Facter 2.0.* 版本在 1.7.* 版本的基础上做了部分的改进与 bug 的修复，另外还增加了一些新的特性，如在之前除返回字符串外，还对整数、浮点数、布尔值、nil（空值）、字符串、数组和哈希做了支持。当 Agent 为微软系列操作系统，通过 operatingsystemrelease 变量，可以收集到更详细的操作系统，如 Windows XP、Windows 2003、Windows 2003 R2、Windows Vista、Windows 2008、Window 7、Windows 2008 R2、Windows 8 和 Windows 2012 等。关于 Facter2.0 版本变化的更多信息，请参考官方网站 http://docs.puppetlabs.com/facter/2.0/release_notes.html。

9.1.2 Facter 参数与应用

下面介绍 Facter 的参数及其应用方法。

1. Facter 参数介绍

由于本书的案例基本使用了 Puppet 2.7 版本，所以我们以 Facter 1.7.4 版本为例来介绍 Facter 命令参数。

```
-y, --yaml
-j, --json
--trace
--external-dir DIR
```

```
--no-external-dir
-d, --debug
-t, --timing
-p, --puppet
-v, --version
-h, --help
```

上面参数的作用如下。

- yaml 参数：通过 yaml 格式向 Facter 导入变量，可以简写为 -y 的形式。
- json 参数：通过 json 格式向 Facter 导入变量，可以简写为 -j 的形式。
- trace 参数：开启后台 traces 功能。此功能通常用于追踪 Facter 调用相关库文件的过程。
- external-dir 参数：导入扩展变量所在的目录或文件。
- no-external-dir 参数：关闭导入扩展变量所在的目录或文件。
- debug 参数：打开调试模式，简写为 -d。以下为 debug 的案例。

通过文件或脚本可以导入 Facter 的 External Facts（在 9.3 节中介绍），这里介绍导入变量失败应该如何的处理。如通过 Shell 脚本导入 Facter 的 External Facts 的方法如下：

```
#!/bin/bash
echo "key1-value1"
```

导入变量失败，可以通过 facter --debug 来查找问题的原因。具体的查找方式如下：

```
...
# 目标文件 /etc/facter/facts.d/sample.txt 不符合语法所以返回了空的数据
Fact file /etc/facter/facts.d/sample.txt was parsed but returned an empty data
set
...
```

以上为 debug 参数输出的结果，通过输出结果我们可以知道问题的原因。之所以会失败，是因为脚本并不符合 Facter 的 External Facts 语法格式。

- timing 参数：显示每个变量的加载时间，单位是毫秒，可以简写为 -t 的形式。
- puppet 参数：加载 Puppet 的库文件，可以简写为 -p 的形式。
- version 参数：输出 Facter 的版本信息，可以简写为 -v 的形式。
- help 参数：输出 Facter 的帮助信息，可以简写为 -h 的形式。

2. Facter 应用

这一小节我们来看一下如何使用 Facter。在系统终端键入 facter，会看到如下输出：

```
# facter
architecture => i386
...
ipaddress => 172.16.182.129
```

```
is_virtual => true
kernel => Linux
kernelmajversion => 2.6
...
operatingsystem => CentOS
operatingsystemrelease => 5.5
physicalprocessorcount => 0
processor0 => Intel(R) Core(TM)2 Duo CPU     P8800  @ 2.66GHz
processorcount => 1
productname => VMware Virtual Platform
...
```

由于输出较多，这里笔者做了截取。通过输出的数据我们可以看到，Facter 收集到的系统信息以 key=>values 作为输出格式，其中 key 为收集到的变量名，它可以在 Puppet 中作为变量使用，value 为收集到的变量值。如以上输出中的 ipaddress => 172.16.182.129，ipaddress 为 key 表示收集 Agent 的 IP，172.16.182.129 为 values 表示收集到的 Agent 具体值。

9.1.3 Facter 与 Puppet 结合

Facter 收集 Agent 信息后，会将这些信息传给 Master，并由 Puppet 语言对数据进行分类与整合，最终根据这些分类后的信息进行差异化配置管理 Agent，这也是 Facter 的主要应用场景。以下为 Puppet 语言中结合 Facter 变量的代码片段。

```
case $::operatingsystem {
  'CentOS': { include centos }
  'MacOS':  { include mac }
}
```

以上代码片就是对收集后的 Agent 信息进行分类处理。其中 operatingsystem 变量值为操作系统的发行版本，我们可以通过“::”方式来调用系统 top 域的变量，也就是 operatingsystem 变量，变量值为 Agent 操作系统发行版本。上述整个代码片段的含义是判断操作系统的发行版本，并根据操作系统的发行版本导入不同的 modules 基础模块文件，最终实现分类管理的功能。

9.2 Facter 常用变量

本节主要介绍 Facter 收集到服务器的变量信息，这里我们以 Facter 1.7.4 版本为例。目前 Facter 1.7.4 版本提供了 94 个变量，由于变量比较多所以这里不全部介绍，只选了一些常用的变量，并将这些变量按功能进行了划分，然后针对划分后的分组分别进行介绍。这里将 Facter 常用的变量分为了 5 组，它们分别是 CPU 相关变量、内存与 swap 相关变量、网络接口与硬件地址相关变量、系统发行版本变量与 kernel 版本相关变量、SELinux 相关变量等。

9.2.1 CPU 相关变量

CPU 是计算机的重要组成部分。冯·诺依曼曾经这样描述：计算机由控制器、运算器、存储器、输入设备、输出设备五大部分组成。运算器指的就是 CPU，随着硬件的发展更新，现代的 CPU 也变得越来越强劲。通过 facter 可获取 Agent 的 CPU 相关的相关变量，通常获取 CPU 的信息用来了解机器的性能，并根据获取到的性能数据对服务器进行差异化配置。以 Web 服务器为例，可以根据收集到 Agent 的性能加载不同的配置文件，以充分利用服务器的性能。

以下为通过 Facter 收集的 CPU 的相关变量。

```
physicalprocessorcount => 2
processorcount => 8
processor0 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
processor1 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
processor2 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
processor3 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
processor4 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
processor5 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
processor6 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
processor7 => Intel(R) Xeon(R) CPU           E5506  @ 2.13GHz
```

通过如表 9-1 所示的变量说明我们可以了解到，Agent 为包含两个物理 CPU、每个物理 CPU 有 4 个核心、累加后共 8 核心的 Intel 处理器的机器。

表 9-1 CPU 相关变量说明

变 量	说 明
Physicalprocessorcount	物理 CPU 个数
Processorcount	两个物理 CPU 的核数累加
processor0-7	每个核 CPU 详细信息

9.2.2 内存与 swap 相关变量

内存与 swap（中文译为“交换分区”或者理解为“虚拟内存”）也是计算机的重要组成部分。还记得我们在第 6 章中介绍的 Puppet 语言数据类型吧？它们均存储在系统的内存中，内存与 CPU 一样也是决定计算机速度的快慢的主要因素。

早期硬件发展的短板导致内存容量不够，为了解决这个问题，操作系统在磁盘开辟了一个区域，这个区域专门用来缓存内存中的数据，这就是 swap 分区。在早期的 UNIX/Linux 发行版本中，供应商总是建议 swap 分区容量要为内存容量的 2 倍。但是伴随硬件的发展，已经没有必要再将 swap 分区容量设置为内存变量的 2 倍了，而是根据机器和机器提供服务器的情况来合理划分 swap 分区容量。不过不管如何划分 swap 分区容量，都建议不要小于 120MB。

下面介绍通过 Facter 收集 Agent 的内存与 swap 分区容量的变量信息，并结合 Puppet 语言来计算系统共使用了多少内存与 swap 分区容量。

以下为 Facter 收集到机器内存与 swap 分区容量的相关的变量。

内存容量如下：

```
memoryfree => 15.16 GB
memorysize => 15.66 GB
```

swap 分区容量如下：

```
swapfree => 1.75 GB
swapsize => 1.92 GB
```

从表 9-2 中我们了解到了上面 4 个变量的具体作用。

表 9-2 内存与 swap 分区相关变量说明

变 量	说 明	变 量	说 明
memorysize	物理内存容量	swapsize	虚拟内存容量
memoryfree	空闲的物理内存容量	swapfree	空闲的虚拟内存容量

下面通过 Puppet 语言来计算机器的实际使用内存与 swap 分区容量的信息。代码片段如下：

```
# 将内存变量中的 GB 单位去掉
$mem_size= regsubst($memorysize," GB","")
# 将空闲的内存变量中的 GB 单位去掉
$mem_free= regsubst($memoryfree," GB","")
# 将 swap 变量中的 GB 单位去掉
$swap_size= regsubst($swapsize," GB","")
# 将空闲的 swap 变量中的 GB 单位去掉
$swap_free= regsubst($swapfree," GB","")
# 计算使用的内存空间
$mem_result = $mem_size - $mem_free
# 计算使用的 swap 空间
$swap_result = $swap_size - $swap_free
# 输出计算后的结果
notify{"memery is :${mem_result}GB and swap is ${swap_result}GB":}
```

如以上代码所示，通过 Facter 获取的机器的内存与 swap 分区容量信息，其中 G 为容量的数量级（1G=1024M）。通过 Puppet 语言计算使用的内存与 swap 已经使用容量。在计算前通过 regsubst 函数将内存与 swap 容量输出的信息进行格式化，符合计算条件后再来计算。最终计算后机器的内存的使用空间为 0.5GB，swap 分区的使用空间为 0.17GB。结果如下：

```
memery is :0.5GB and swap is 0.17GB
```

随着我们对 Puppet 的深入学习，更好的应用方式是根据机器配置的不同，来动态加载服务的配置文件。如 Apache、Nginx、Squid 和 Memcached 等，根据 Agent 的内存与 sawp 使用情况来动态加载配置，从而提高 Agent 的内存与 sawp 的利用率，降低服务器的成本。以 Memcached 为例，使用当前剩余内存的 80% 来启动 Memcached 的守护进程。

```
# 为了方便内存间差值的数学运算，通过 regsubst 函数将 Facter 内存变量中的 GB 单位替换为空
$mem_free= regsubst($memoryfree," GB","")
# 计算 80% 内存大小
$mem = $mem_free * 0.8
# 通过 exec 资源启动自定义参数的 Memcached 守护进程
exec { 'memcached':
  command     => "/usr/bin/memcached -d -m $mem -u guest -p 12111 ",
  refreshonly => true,
}
```

9.2.3 网络接口与硬件地址相关变量

网络接口与硬件地址的作用是帮助机器与外界建立联系，建立联系的方式包括连接外部的国际互联网与企业内部网等。下面介绍通过 Facter 来收集 Agent 的网络接口、接口 IP 与硬件地址等变量。在之前章节介绍的知识中了解到 Puppet 通过 node 节点来管理服务器，但偶尔也需要更细的分类与管理，这时就可以通过收集的 Agent 网络接口与接口 IP 的信息进行更精细的配置管理。

以下为 Facter 收集的机器网络接口变量。

```
interfaces => eth0,eth1,ip6tnl0,sit0,tunl0
```

通过 interfaces 变量的值可以了解到本机共有 5 个网络接口。

以下为 Facter 收集的网络接口 IP 与子网掩码等变量。

```
ipaddress => 192.168.1.7
netmask => 255.255.255.128
ipaddress_eth1 => 192.168.1.8
netmask_eth1 => 255.255.255.128
```

ipaddress 变量中存放本机的 IP 地址，netmask 变量中存放本机的子网掩码变量，ipaddress_eth1 与 netmask_eth1 为网络接口 eth1 的 IP 与子网掩码。注意，facter 只显示网络接口有 IP 的情况。

以下为 Facter 收集的网络接口的网卡硬件地址变量。

```
macaddress => 08:19:A4:27:6D:99
macaddress_eth0 => 08:19:A4:27:6D:99
macaddress_eth1 => 08:19:A4:27:6D:1A
```

网卡硬件地址又称 MAC 地址，它长 48bit，由 12 位的十六进制数字组成，其中 0 到 23 位是厂商向 IETF 等机构申请的表示厂商的代码。MAC 地址也是网卡的唯一标识变量。

facter 除了支持收集 IPv4 版本地址外，还支持收集 IPv6 的变量，默认会将 IPv6 变量存放于 $ipaddress6 和 $ipaddress6_{NETWORK INTERFACT} 两个变量中。

补充 当处于一个复杂的网络环境时，如主机连接路由器，再从路由器连接服务器，这时主机获取到的并非服务器的真实 MAC 地址，而是路由器的 MAC 地址，而通过 facter 就可以抓取到最原始的 MAC 地址。

9.2.4 系统发行版本变量与 kernel 版本相关变量

下面介绍系统的发行版本相关变量与 kernel 版本相关的变量。

1. 系统的发行版本相关变量

以下为 Facter 收集到的系统发行版本和版本号等变量。

```
operatingsystem => SLES
operatingsystemrelease => 10.1
```

表 9-3 操作系统发行版本变量说明

变 量	说 明
Operatingsystem	操作系统发行版本
Operatingsystemrelease	操作系统发行版本号

通过表 9-3 我们了解到机器的发行版本为 Suse，版本号为 10.1。

我们再来看 operatingsystem 变量与 Puppet 结合的案例。这个案例实现的是通过判断系统的发行版本来安装不同的 Apache 包。代码片段如下：

```
$packages = $operatingsystem ? {
    /(?i-mx:ubuntu|debian)/        => 'apache2',
    /(?i-mx:centos|fedora|redhat)/ => 'httpd',
                         default => 'httpd'
}
package { $packages:
 ensure => install,
}
```

如以上代码所示，根据 operatingsystem 操作系统的发行版本安装不同的软件包，如果是 Ubuntu 或 Debian 系统发行版本则安装 apache2 软件包；如果是 CentOS、Fedora 或 RedHat 系统发行版本则安装 httpd 软件包；如果都不匹配则安装默认的 httpd 软件包。

2. kernel 版本相关的变量

kernel（中文译为“内核”）是操作系统的一个重要组成部分，它主要与硬件打交道。这里 Facter 可以收集到 kernel 的版本等相关信息，并根据收集到的 kernel 版本信息来安装配置软件，有效地避免软件安装过程中操作系统层面容易出现的一些问题，如驱动不兼容、程序版本与内核版本不兼容等。

以下为 Facter 收集的 kernel 内核版本等相关信息。

```
kernel => Linux
kernelmajversion => 2.6
```

```
kernelrelease => 2.6.32-279.el6.x86_64
kernelversion => 2.6.32
```

从表 9-4 中我们了解到，机器是 Linux 操作系统，其使用 kernel 的 2.6 版本。

表 9-4 Kernel 变量说明

变 量	作 用	说 明
kernel	内核名	微软系列系统返回 windows 其他系统返回 uname –s
kernelmajversion	大版本号	返回操作系统版本号
kernelrelease	完成版本号	微软系列系统返回 Win32_OperatingSystem AIX 系统返回 oslevel -s 其他系统返回 uname –r
kernelversion	小版本号	Solaris 系统返回 uname -v Sunos 系统返回 uname -v 其他系统返回 kernerlrelease

9.2.5 SELinux 相关变量

SELinux 对我们来说都不陌生，它是安全增强式的 Linux（SELinux 全称为 Security-Enhanced Linux），是 MAC（Mandatory Access Control，强制访问控制系统）的一个实现。目的在于明确指出某个进程可以访问哪些资源，如文件、网络端口等。强制访问控制系统的用途在于增强系统抵御 0-Day 攻击（利用尚未公开的漏洞实现的攻击行为）的能力，是一种强制访问控制的实现，所以 SELinux 足够安全，安全到让我们对操作系统的正常需求变得更加的复杂甚至出现莫名其妙的问题，所以导致我们经常尝试关闭它。SELinux 主要由美国国家安全局开发，并于 2000 年 12 月 22 日发布给开放源代码的开发社区。以下为 facter 程序获取 Agent 的 SELinux 变量信息，我们可以通过收集到的信息并根据服务器的重要程度来确认是否要配置开启或关闭 SELinux 功能。

以下为 Facter 收集的 SELinux 变量。

```
selinux => true
selinux_config_mode => enforcing
selinux_config_policy => targeted
selinux_current_mode => enforcing
selinux_enforced => true
selinux_mode => targeted
selinux_policyversion => 24
```

从参数（见表 9-5）返回的变量可以了解到，目前机器已经开启了 SELinux 机制。如果读者所在的企业网足够安全，或者读者足够了解所在网络的安全程度的情况下，建议关闭 SELinux 功能。

表 9-5 SELinux 相关变量说明

变 量	说 明
selinux	Agent 是否启用 SELinux
selinux_config_mode	SELinux 配置模式，包含 enforcing、permissive 和 disabled
selinux_config_policy	SELinux 的规则配置
selinux_policyversion	SELinux 的版本
selinux_enforced	SELinux 开关状态

9.3 扩展 Facter

本节介绍如何扩展 Facter 的 facts（其实 facts 也是变量，所以下文将 facts 统称为变量），由于篇幅所限，这里仅对扩展 Facter 进行简单介绍，关于其本身扩展模块的更多信息可以参考 http://downloads.puppetlabs.com/facter/apidocs/。

9.3.1 扩展 Facter 的变量

以 1.7 版本为例，Facter 为我们提供了 94 个变量，当这些变量不能满足我们的需求或者希望通过 Facter 来收集一些定制化的变量时，就需要去扩展 Facter。扩展 Facter 并不复杂，它包含了以下 3 种常见的扩展方式：

- 通过 Ruby 语言来扩展 Facter 的变量，适用于了解 Ruby 语言的用户人群。
- 通过环境变量来扩展 Facter 的变量，适用于临时的简单需求用户人群。
- 通过 External Facts 方式扩展 Facter 的变量（此功能仅限于 Facter 1.7 以上版本），适用于利用不同编程语言来扩展 Facter 变量的用户人群。

1. Ruby 语言来扩展 Facter 的变量

与 Puppet 一样，Facter 也是由 Ruby 语言开发的，所以这里介绍如何通过 Ruby 语言来扩展 Facter 变量。这里需要读者掌握一些 Ruby 语言的基本语法，但是不用担心，因为 Ruby 语言并不难学。下面介绍如何通过 Ruby 语言来扩展 Facter 的变量。首先介绍扩展变量的程序目录位置在哪，然后再来通过两个小的案例扩展 Facter 的变量。

先来介绍 Facter 的扩展程序目录位置。通过 facter | grep rubysitedir 命令可以查到 Ruby 的存放位置，Facter 扩展程序的位置就在 Ruby 的目录下。这种方法也适用于源码编译安装与 yum 方式安装。两种安装方式路径如下：

```
rubysitedir => /usr/local/lib/ruby/site_ruby/1.8        # 源编译方式 Ruby 的位置
rubysitedir => /usr/lib/ruby/site_ruby/1.8              #yum 安装方式 Ruby 的位置
```

以源码编译 Ruby 方式为例，扩展 Facter 程序的目录位置就在 /usr/local/lib/ruby/site_ruby/1.8/facter/ 目录中。进入此目录我们会发现有很多的 *.rb 文件，如图 9-1 所示。这些都

是 Facter 变量的源程序文件，由 Ruby 语言编写。如果读者对 Ruby 语言比较熟悉，这里的源程序也可以作为参考，通过对默认变量源码的分析与学习，可加深对 Facter 的了解。

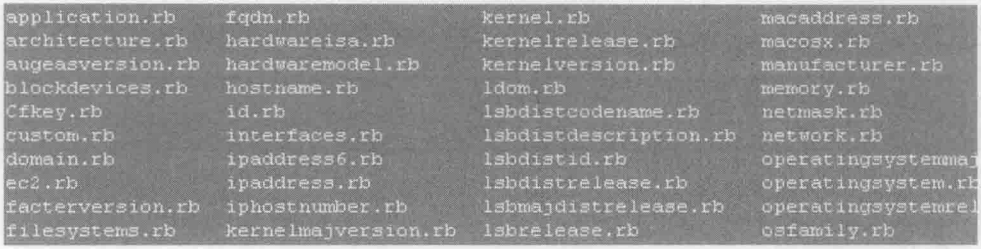


图 9-1 Facter 默认变量源程序

这里来介绍两个扩展 Facter 的案例。

案例 1

通过 Ruby 语言来扩展一个 Facter 的变量。编辑 /usr/local/lib/ruby/site_ruby/1.8/facter/custom.rb 文件，将以下代码追加到文件中。

```
Facter.add(:custom)do
  setcode do
    "This is custom facter"
  end
end
```

如以上代码所示，其中 :custom 为 Facter 中的 key，This is custom facter 为 Facter 程序的 values。保存 custom.rb 文件后执行 facter，以下为输出的结果：

```
facter | grep custom
custom => This is custom facter
```

案例 2

通过 Ruby 语言调用系统命令，并将命令值作为 Facter 的变量（Ruby 语言区分大小写）。编辑 /usr/local/lib/ruby/site_ruby/1.8/facter/ loadavg.rb 文件，将以下代码追加到文件中。

```
Facter.add(:loadavg) do
  setcode do
    Facter::Util::Resolution.exec('cat /sys/loadavg')
  end
end
```

如以上代码所示，其中 loadavg 为 Facter 中的 key，Facter::Util::Resolution.exec('cat /sys/loadavg') 调用系统命令，并将系统命令的返回值存入 loadavg 值中，作为 loadavg 变量的值。

```
facter | grep loadavg
loadavg => 0.03 0.07 0.08 1/184 20415
```

Facter扩展变量的程序默认放在/usr/lib/ruby/site_ruby/1.8/facter/目录下。还可以通过环境变量的方式来增加或改变扩展变量的存放目录。如追加/home/fact目录为扩展变量程序目录，将loadavg.rb程序移动到此目录下，并追加此目录的环境变量。操作流程如下：

```
# mkdir -p /home/fact
# mv /usr/lib/ruby/site_ruby/1.8/facter/loadavg.rb /hoem/fact/
$ export FACTERLIB="./hoem/fact"     # 导入扩展变量目录到 Facter 环境变量中
facter | grep loadavg
loadavg => 0.03 0.07 0.08 1/184 20415
```

2. 通过环境变量来扩展 Facter 的变量

如果没有学过编程语言或者觉得学习Ruby语言比较痛苦，还可以通过加载Facter的环境变量的方式来扩展变量。导入格式如下：

```
export Facter_"变量名"= "变量值"
```

以下为通过环境变量方式扩展Facter的变量的案例，将系统负载信息通过环境变量的方式导入Facter。

```
export FACTER_loadavg=`cat /sys/loadavg` # 将 cat 命令输出的结果通过环境变量方式导入 Facter
#facter | grep loadavg                   # 再次执行 facter 会看到通过环境变量方式导入的系统负载值
loadavg => 0.03 0.07 0.08 1/184 20415
```

9.3.2 External Facts 外部扩展变量

External Facts（扩展变量，下称扩展变量），为Facter 1.7.*新增加的功能。它提供了一个途径让我们可以将文件或者脚本编程语言按照Facter的指定格式导入Facter中作为变量使用。在使用扩展变量前，首先介绍存放这些文件和脚本编程语言在系统的位置，如表9-6所示。在不同的操作系统发行版本中扩展变量的目录是不一样的。

表9-6 扩展变量默认目录

操作系统发行版本	说 明
UNINX/Linux/MAC	/etc/facter/facts.d/ # 开源社区版本标准目录 /etc/puppetlabs/facter/facts.d/ # 开源社区版本标准目录
Windows 2003	C:\Documents and Settings\All Users\Application Data\PuppetLabs\facter\facts.d\
微软其他的操作系统（Windows Vista、Windows 7\ 8\ 2008\2012等）	C:\ProgramData\PuppetLabs\facter\facts.d\

当我们将文件或脚本编程语言放入扩展变量的默认目录后，Facter本身会自动解析默认目录内的文件或者脚本编程语言的输出值，并将它们导入Facter变量中来使用。


注意 扩展变量目前只支持 Facter 1.7 以上的版本，在使用前请通过 facter -v 确认自己的版本。

除了通过默认目录来扩展 Facter 变量外，还可以通过 Facter 的 external-dir 参数指定目录方式来导入变量。通过 external-dir 导入方式如下：

```
facter --external-dir=/home/facts/ | grep key
key1 => value1
key2 => value2
key3 => value3
key4 => value4
key5 => value5
```

可以看到 Facter 程序输出中多了很多自定义的变量值，这些变量均通过 /home/facts/ 目录中的文件或者脚本输出得到。这里可以使用的文件或者脚本包括文本文件、结构化文件、批处理脚本、Python 脚本和 PHP 脚本等。Facter 的扩展变量支持以下的导入格式：

```
key1=value1
key2=value2
key3=value3
```

通过其他编程语言或者结构化文件导入扩展变量都需要遵循 Facter 的扩展变量的格式。下面笔者通过 Python 脚本、PHP 脚本、微软的 bat 程序和结构化文件（包含 json 和 yaml）方式来实现导入 Facter 的扩展变量。

1. Python 脚本的扩展变量

可以通过 Python 脚本来导入扩展变量。以 UNIX/Linux/MAC 为例，在 Facter 的扩展变量默认（/etc/facter/facts.d/）目录中编辑 my_fact_script.py 脚本。代码片段如下：

```
#!/usr/bin/env python
data = {"key1" : "value1", "key2" : "value2" }
for k in data:
      print "%s=%s" % (k,data[k])
```

编写后不要忘记为 my_fact_script.py 文件设置权限，如下：

```
chmod +x /etc/facter/facts.d/my_fact_script.py
```

通过 Python 测试程序的最终输出格式如下：

```
# python my_fact_script.py
key1=value1
key2=value2
key3=value3
```

这时我们可以执行 Facter 程序，确认 Python 脚本导入的变量是否已经成功导入。

2. PHP 脚本的扩展变量

可以通过 PHP 脚本导入扩展变量。还是以 UNIX/Linux/MAC 为例，在 Facter 的扩展变量默认（/etc/facter/facts.d/）目录中编辑 my_fact_script.php 文件。代码片段如下：

```
#!/usr/bin/php
<?php
   $arr=array(     # 定义 php 的 $arr 数组并追加内容
              "key1"=>"value1",
              "key2"=>"value2",
              "key3"=>"value3",
              "key4"=>"value4",
              "key5"=>"value5",
   );
   foreach($arr as $k=>$v){     # 循环便利 $arr 数组生成 Facter 所需要的格式
              $output=sprintf("%s=%s",$k,$v);
              echo $output."\n";
   }
?>
```

执行 my_fact_script.php 文件输出结果如下：

```
#./my_fact_script.php
key1=value1
key2=value2
key3=value3
```

这时可以执行 Facter 程序，确认 PHP 脚本导入的变量已经成功导入。值得注意的是可以将 Facter 的扩展变量存放到数据中，并与脚本编程语言结合将数据库中的内容作为 Facter 的变量，这样使得扩展 Facter 的变量变得更加灵活方便。

3. Windows 系统导入扩展变量

微软系列的操作系统支持导入扩展变量包含 exe 和 com 两种可执行文件的方式，还包含 .bat 和 .cmd 两种脚本文件方式。以 Windows 7 系统为例，在 C:\ProgramData\PuppetLabs\facter\facts.d\ 目录下，编辑 my_fact_script.bak 文件，内容如下：

```
@echo off
echo key1=val1
echo key2=val2
echo key3=val3
REM Invalid - echo 'key4=val4'
REM Invalid - echo "key5=val5"
```

此方式仅限于在微软 Windows 系列操作系统中使用。

4. 文件与结构化文件导入扩展变量

扩展变量还支持文件与结构化文件的导入。

文本文件格式导入如下：

```
key1=value1
key2=value2
key3=value3
```

yaml 格式导入如下：

```
---
key1: val1
key2: val2
key3: val3
```

json 格式导入如下：

```
{
  "key1": "val1",
  "key2": "val2",
  "key3": "val3"
}
```

以上为文本文件、yaml 格式和 json 格式导入 Facter 扩展变量的方法。通过以下测试数据来看，以文本文件扩展 Facter 变量方式解析速度最快，但扩展性没有脚本编程语言方便。

```
# facter --timing
kernel: 14.81ms
/usr/lib/facter/ext/abc.sh: 48.72ms
/usr/lib/facter/ext/foo.sh: 32.69ms
/usr/lib/facter/ext/full.json: 104.71ms
/usr/lib/facter/ext/sample.txt: 0.65ms    # 文本文件扩展 Facter 变量方式（单位为毫秒）
```

结构化文件在微软的 Windows 系列系统中，需要注意以下两点：

- 结尾可以使用 LF 或 CRLF 换行符。
- 本件编码必须使用 ANSI 或 UTF8 编码。

9.4 编写与分发 Facter 的扩展

前面我们介绍了如何在 Master 上扩展 Facter 的变量。本节来介绍一下如何将这些扩展变量由 Master 同步到 Agent 上。以分发“收集 Agent 的负载信息”程序为例来介绍。从 Master 分发 Facter 扩展到 Agent 共分 4 步：

步骤 1　追加以下配置到 Master/Agent 的 puppet.conf 文件中，其含义是打开 Puppet 的插件同步开关。

```
[main]
pluginsync = true
```

步骤 2 将以下代码片段追加到 /etc/puppet/modules/facter/lib/facter loadavg.rb 文件中。

```
Facter.add("loadavg") do
 setcode do
      # 需要对"'"进行转义
      Facter::Util::Resolution.exec('uptime | awk \'{print $(NF-2)}\' | sed \'s/,//g\'')
 end
end
```

需要注意，Ruby 语言是区分大小写的，它在调用系统命令时需要对特殊符号进行转义。loadavg.rb 文件的作用是通过机器的 uptime 命令来收集当前机器的负载状况，并将负载状况赋值给 loadavg 作为 Facter 的输出。追加后测试结果如下：

```
# facter | grep loadavg
loadavg => 0.11
```

步骤 3 步骤 1 和步骤 2 完成后，已经成功地在 Master 服务器上通过 loadavg.rb 扩展了 Facter 的 loadavg 变量，并且打开了同步的配置选项。下面再来看如何将 loadavg.rb 同步到 Agent 上。在介绍同步方法前，先介绍一下 Master 的相关扩展目录结构与含义。

```
{modulepath}
└── {module}
    └── lib
        ├── facter
        └── puppet
            ├── parser
            │   └── functions
            ├── provider
            └── type
```

上边目录结构的含义如下。

- modulepath：定义在 puppet.conf 文件中，表示 modules 发布目录的位置。
- module：模块的名字。
- lib：库文件存放目录。
- facter：Facter 的扩展源程序存放目录。
- functions：Puppet 自带函数扩展源程序存放目录。
- provider：Puppet 的提供者扩展源程序存放目录。
- type：Puppet 的扩展源程序存放目录。

以 facter 模块为例，以下为 Master 扩展目录的路径。将 loadavg.rb 文件复制到 /etc/puppet/modules/facter/lib/facter/ 目录下。注意插件同步发生在 Puppet 运行的阶段，即添加插件后当 Agent 首次访问 Master 时，这时的 loadavg.rb 插件是不能使用的，首次只会将 loadavg.rb 插件文件同步到 Agent 的目录上。当下次 Agent 再次访问 Master 时，loadavg.rb 扩展的变量才可以使用。

步骤 4　在 Agent 确认 puppet.conf 文件已经开启了 pluginsync = true 配置后，再执行以下命令：

```
# puppet agent --server puppet.example.com  --test
```

以下为输出结果：

```
info: Retrieving plugin
notice: /File[/usr/local/puppet/lib/ruby/site_ruby/1.8/facter/loadavg.rb]/ensure:
defined content as '{md5}9213ddc3bc2a7c9778ea552feecda140'
# 成功的加载 loadavg.rb 文件
info: Loading downloaded plugin /etc/puppet/modules/facter/lib/facter loadavg.rb
info: Loading facts in /usr/local/puppet/lib/ruby/site_ruby/1.8/facter/loadavg.rb
```

从输出结果中可以看到，loadavg.rb 文件已经成功地由 Master 同步到 Agent 的 /usr/local/puppet/lib/ruby/site_ruby/1.8/facter/ 的目录中，在 Agent 执行以下命令，可以看到 Agent 的相关负载状况。

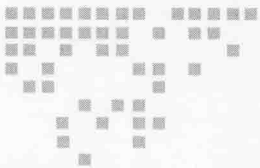
```
# facter | grep loadavg
loadavg => 0.11
```

当 Agent 再次访问 Master 时也会将 Agent 的负载信息传入 Master 中。

第三部分 Part 3

高 级 篇

- 第10章 Puppet高级功能
- 第11章 Puppet集群技术
- 第12章 报告系统
- 第13章 Puppet Web GUI
- 第14章 PuppetDB数据仓库
- 第15章 Marionette Collective框架应用



Chapter 10 第 10 章

Puppet 高级功能

本章主要介绍 Puppet 的一些高级功能，通过对本章的学习有助于我们更深入地了解 Puppet。本章首先介绍 ENC 的功能，之前知识体系中所掌握的通过清单来管理配置服务器，而 ENC 则是清单配置管理的一个替代工具，它的优势是通过不同的语言转为 YAML 格式来管理配置服务器，弥补了 Puppet 清单管理配置服务器的一些不足，从而更适用于管理海量服务器场景。接着介绍 Puppet DSL，Puppet 通过 DSL 继承了 Ruby 语言的全部功能，并可以通过 Ruby 语言与 Puppet 结合管理服务器。然后介绍 Puppet 的关系图，Puppet 本身可以生成 DOT 语言，可以通过 DOT 语言与工具的结合描绘出 Puppet 代码之间的逻辑关系，从而避免了逻辑复杂导致的配置错误。接着介绍 Puppetlabs-stdlib，它是 Puppet 官方网站提供的集成工具包，包含了自定义函数、资源与提供者的源码与使用案例，主要介绍它丰富的函数功能，并为后一节开发 Puppet 的扩展作铺垫，了解 Puppetlabs-stdlib 使用与代码逻辑可为后续独立开发扩展打下良好基础。最后将介绍 Puppet 的扩展开发，从代码结构中了解 Puppet 是如何工作的，以及为什么通过资源可以在不同的发行版本机器上安装配置软件。

10.1 ENC 介绍

在前面章节介绍的知识体系中，管理配置服务器通常使用 manifests 中的清单来完成，这里我们再来了解另一种配置管理服务器的方式——ENC（External Node Classifiers，外部节点分类器）。ENC 是 Puppet 中的脚本抽象层，它可以通过 Ruby、Shell、PHP、Perl 和 Python 等脚本语言来替代 manifests 中的清单功能，但要求这些编程语言能够正确地输出

YAML 标记语言格式。也正是因为与编程语言的结合，使得 ENC 通过 Puppet 管理配置服务器变得更加灵活，并适用于海量服务器配置管理的场景。目前 Google 与 Zynga 等公司部分服务均利用 Puppet 的 ENC 功能协助管理海量的节点。另外 Puppet 还可以通过 LDAP 服务作为节点分类器，不过由于笔者工作中并未涉及 LDAP，所以这里不会介绍。关于 LDAP 配置的更多信息请参考官方网站 http://docs.puppetlabs.com/guides/ldap_nodes.html。

10.1.1 ENC 的配置

因为 ENC 功能是通过编程语言来生成 YAML 格式完成整个配置环节的，所以在正式介绍如何配置 ENC 前，首先来了解一下什么是 YAML 格式，YAML 格式与常用的 XML 格式有什么区别。

1. YAML 格式介绍

YAML（Yet Another Markup Language）是一种标记语言。YAML 设计的初衷如下：

- 方便人们阅读。
- 适合描述程序语言的数据结构。
- 用于不同程序间的数据交换。
- 支持泛型工具。
- 支持串行处理。
- 丰富的表达能力和可扩展性。
- 易于使用。

YAML 以哈希表的方式来表达自己。笔者以本书为题介绍一个简单的例子，帮助读者了解 YAML 是如何工作的。

```
1  # 开始
2  ---
3  书名   : 'Puppet 权威指南'
4  出版社 : '机械工业出版社'
5  作者   : '远航'
6  前 1 章  :
7  - 第 1 节 : 优秀运维工程师 vs 普通运维工程师
8  - 第 2 节 : 自动化运维工具箱
9  前 2 章  :
10 - 第 1 节 : DevOps 介绍
11 # 结束
```

以上 YAML 的例子中，相关符号含义如下：

- “#”表示注释。
- “---”表示 YAML 的开始。紧接着“书名”作为键，通过“:”进行分割，“Puppet 权威指南”为值，其余以此类推。

- 第 7 行中的符号“-”表示第 1 节为第 1 章的列表成员，“优秀运维工程师 vs 普通运维工程师”为第 1 节的值，并以此类推。

这些符号最终组成了 YAML 的语法格式。需要强调的是：YAML 语法规范很重要，如果编写后的 YAML 不能肯定语法是否正确，可以通过 http://YAML-online-parser.appspot.com/ 在线编辑器来确认，以降低程序读取配置出错的概率。

2. YAML 格式与 XML 格式的比较

YAML 与 XML 是两个常见的使用格式，两者相比有什么差别呢？与 YAML 相比，XML 关注面比较多，可以说是面面俱到。XML 是一个典型的由委员会驱动的“庞然大物”，它试图成为一种文档格式、数据格式、消息包格式、安全的 RPC 通道（SOAP），以及一种对象数据库。而且，XML 为每一类型的访问和操作都提供了大量的 API，包括 DOM、SAX、XSLT、XPATH、JDOM，以及许多不太常见的接口层。XML 完成了所有的这些工作，这非常了不起；但令人失望的是没有一项工作是完美无缺的。

与 XML 相比 YAML 关注面比较窄，它只是清晰地表示在动态编程语言（如 Perl、Python、Ruby 和 Java 语言等）中所遇到的数据结构以及数据类型。目前，对于这些语言，已经有了一些绑定类库。另外许多其他的编程语言中已存在可以很好地使用 YAML 的数据模型，但目前还没有人编写相关的类库，这些语言包括 Lisp/Scheme、Rebol、Smalltalk、xBase 和 AWK 等。

3. ENC 配置

Puppet 如何开启 ENC 功能呢？在 Master 上编辑 /etc/puppet/puppet.conf 配置文件，将以下内容追加到配置文件的 master 段中。

```
[master]
node_terminus = exec
external_nodes = /etc/puppet/enc/ruby_enc.rb
```

代码中参数作用如下。

- master 参数：在第 4 章中介绍过，puppet.conf 主要分为 3 个段，其中 [master] 段用于 Master 部分的配置。[master] 段的参数主要应用于守护进程的配置。
- node_terminus 参数：选择 catalog 的编译方式。目前 Puppet 提供了 3 种 catalog 的编译方式，它们分别为默认的 plain 方式和可选的 puppet apply 方式与 exec 执行 ENC 脚本方式。plain 方式主要用于 Puppet 的 C/S 架构场景，即 Agent 与 Master 不在同一台服务器，Agent 通过网络连接 Master 并获取 catalog；puppet apply 方式适用于单机执行 Puppet 代码，但需要注意，puppet apply 方式不支持 Puppet 代码中节点的继承关系（笔者通常使用此方式测试资源的最终执行结果）；最后是 exec 即 ENC 脚本方式，它通过不同的语言生成统一的 YAML 格式，并根据 YAML 格式通过

Puppet 编译成 catalog。在此设置为 exec 方式，通过它开启 ENC 的配置开关。

- external_nodes 参数：用于设置 ENC 脚本的路径与脚本名。如果 /etc/puppet/enc 路径不存在，则需要手工创建它。ruby_enc.rb 为 ENC 的脚本，这里可以使用 Shell、PHP、Ruby、Perl 和 Python 等常见的编程语言来编写。

配置 puppet.conf 文件变更后，需要重启 Master 的守护进程，ENC 功能才会生效。

开启 ENC 功能后，manifests 目录中的清单功能使用级别将会降低，Agent 访问 Master 时会优先匹配 ENC 功能。如果希望既使用 manifests 目录中的清单功能，又使用 ENC 的功能，则 ENC 需要返回的是空的 YAML，Agent 会自动匹配 manifests 目录中的清单功能。另外在使用 ENC 功能时还需要注意 manifests 与 ENC 功能的区别。

- 在 manifests 管理方式中，支持的 Agent 的 Hostname 方式包括 agent1.example.com、agent1.example（简写）和 agent1（简写），但是在 ENC 中不支持这样的简写 Hostname 方式，需要写域名的全称，即 agent1.example.com。
- 在 manifests 管理方式中，Agent 会依次匹配 site.pp 文件中的 node，如果没有匹配到 node，则最终会落到 node default 节点上。但是在 ENC 节点配置中没有默认节点的管理方式。

10.1.2 ENC 案例

本小节将以 Ruby 语言为例来介绍 ENC 的使用案例。之所以选择 Ruby 语言，是因为 YAML 与 Ruby 联系非常紧密，在很多 Ruby 项目中都使用了 YAML 格式来保存配置文件。毫不夸张地说，YAML 是 Ruby 中“流动的血液”。

在确认打开 ENC 功能后，应首先判断 puppet.conf 配置文件中的 ENC 功能是否打开，并且 external_nodes 参数指定的脚本返回正确的 YAML 格式，如果返回的是正确的 YAML 格式，则加载 ENC 的配置；如果 puppet.conf 文件中 ENC 功能并没有打开或者它返回的是空的 YAML 格式，则加载 manifests 清单目录中的 site.pp 文件，如图 10-1 所示。

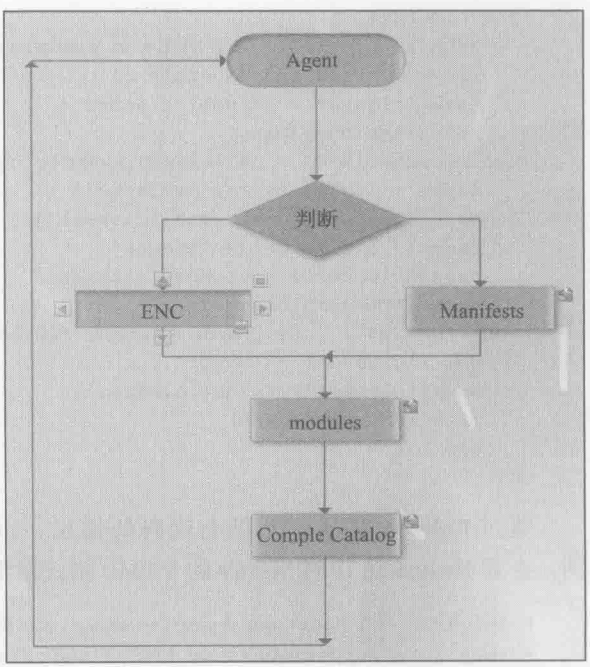


图 10-1 ENC 流程图

确认 Master 的 puppet.conf 配置文

件中开启了 ENC 配置后，追加以下内容到 /etc/puppet/enc/ruby_enc.rb 脚本文件中。

```
#!/usr/local/bin/ruby
# encoding: UTF-8
# filename:puppet_enc.rb
# time:2014.01.01
# 参数入口
node = ARGV[0]
# 通过正则匹配输入的 node 节点是否合法，目前只支持 web.example.com | cache.example.com | db.example.com 节点
unless node =~ /^(web|cache|db)\.(\S+\.\S+)$/
    print "please input web.example.com | cache.example.com | db.example.com\n"
    exit 0
end
# 加载 YAML 的库
require 'yaml'
# 加载节点
hostname = $1
# 生成 Puppet 的清单列表
default = {'classes' => []}
base = { 'enviroment' => 'production',
    'parameters' => {
        'puppetserver' => 'puppet.example.com'
    },
    'classes' =>[ 'base'],
}
case hostname
    when /^web?\w+$/    # 匹配 web.example.com 节点，并加载 base 与 nginx 类
    web = { 'classes' => 'nginx'}
    base['classes'] << web['classes']
    puts YAML.dump(base)
when /^cache?\w+$/    # 匹配 web.example.com 节点，并加载 apache、memcache 类
    cache = {'classes' =>'memcache'}
    base['classes'] << cache['classes']
    cache = {'classes' =>'apache'}
    base['classes'] << cache['classes']
    puts YAML.dump(base)
when /^db?\w+$/    # 匹配 db.example.com 节点，并加载 mysql 类
    db= {'classes' =>'mysql'}
    base['classes'] << db['classes']
    puts YAML.dump(base)
    end
exit 0
```

通过 Ruby 来测试一下以上代码的输出。以 Agent 的 Hostname 为 web.example.com 为例，查看 Hostname 访问 Master 的 YAML 格式输出，看 ENC 是如何加载配置的。具体如下：

```
# 以下为 ENC 根据 Hostname 为 web.example.com 返回的 YAML 格式配置
# ruby /etc/puppet/enc/ruby_enc.rb web.example.com
---
```

```
enviroment: production
parameters:
 puppetserver: puppet.example.com
classes:
- base
- nginx
```

如果我们将以上的ENC功能输出内容转换为manifests中的清单代码，它与以下清单中的代码功能一致。

```
node web.example.com {
     $puppetserver = puppet.example.com
     include base
     include nginx
}
```

整个Agent访问Master的工作流程为：当Agent的Hostname为web.example.com访问Master时，Master会将Agent的Hostname（即web.example.com）作为ENC的ruby_enc.rb脚本输入。ruby_enc.rb脚本执行过程中，首先会检查Agent的Hostname（web.example.com）是否符合规范（目前此脚本只接受Hostname为web.example.com、cache.example.com和db.example.com的域名访问），当检查通过后，接着来自Hostname（web.example.com）的Agent会加载$puppetserver变量以及base和nginx模块，这些模块与manifests的清单功能一样，都存放在modules目录中。最后根据模块中的配置信息在Agent上应用它们，这就是ENC的整个工作流程。从以上的案例中可以看到，ENC主要承接了manifests中的逻辑等功能，manifests的缺点是逻辑与数据结合不够灵活，而ENC恰好相反，它使用了编程语言，以Ruby语言为例，通过Ruby语言弥补了Puppet的一些不足，如循环的缺失、连接数据库的限制等，而这一切都可以通过ENC很好地解决。

10.2 Ruby DSL介绍

Ruby DSL（Ruby Domain Specific Language）是一个面向语言的工具，用于解决某个特定领域的编程任务，Puppet通过Ruby DSL功能继承了全部的Ruby语言能力。Puppet 2.6版本到2.7支持Ruby DSL功能，可惜的是Puppet 3中已经取消了对Ruby DSL的支持，不过对于使用2.7的用户来说还是可以了解一下Ruby DSL这一工具的。

应该在什么时候使用Ruby DSL呢？如通过Puppet ENC的API来访问一个动态数据集合时，Ruby DSL功能可以完美地解决数据资源声明的问题。另外Ruby中使用了each方法来遍历哈希表和数组，each方法提供了声明多资源的简便方式，而这一工作我们想通过Puppet语言来解决是很困难的。在manifests和modules中既可以使用Puppet语言也可以使用Ruby DSL。它们之间的区分可以通过扩展名来识别，Ruby DSL以.rb结尾作为扩展名，

Puppet 语言以 .pp 结尾作为扩展名。使用 Ruby DSL 还有以下 3 个优势：

- Ruby DSL 可以让开发人员直接生成 Puppet 的资源。
- Ruby DSL 可以克服 Puppet 的限制。
- Ruby DSL 可以借助 Ruby 强大的单元测试框架。

不过需要注意的是 Ruby DSL 在 Puppet 中还是 Puppet DSL 的一个子集，因此在使用 Ruby DSL 时还会有一些限制，如在使用 Ruby 声明类时，不支持 Puppet 2.6 中运行阶段的 stages 特性。

10.2.1 如何使用 Ruby DSL

如何通过 Puppet 使用 Ruby DSL 功能呢？看下面这个例子。首先编辑 /etc/puppet/dsl/file.rb 文件，将以下代码片段追加到文件中。

```
# 创建默认的节点
node 'default' do
  # 调用 file 资源创建 test 文件
  file('test',     # 设置资源标题
    :ensure => 'file',     # 设置创建文件类型
    :path => '/tmp/test',     # 设置创建文件路径
    :mode => '0640')     # 设置文件的权限
end     # 结束符
```

执行（puppet apply /etc/puppet/dsl/file.rb）以上的 file.rb 代码片段，结果如下：

```
notice: /Stage[main]//Node[default]/File[/tmp/test]/ensure: defined content as
'{md5}37b51d194a7513e45b56f6524f2d51f2'
notice: Finished catalog run in 0.04 seconds
```

从 Ruby DSL 的输出结果中可以看到，Ruby DSL 在 /tmp/test 下创建好了文本文件，并将 test 追加到文件中。Ruby DSL 不但可以使用 Puppet 资源，还继承了 Puppet 的很多功能。继续看两个 Ruby DSL 的案例。

10.2.2 Ruby DSL 案例

本小节通过两个案例具体介绍 Ruby DSL 如何与 Puppet 的结合来管理 web.example.com 站点。

案例 1 通过 Ruby DSL 批量创建系统账户

首先配置一下 Ruby DSL 的环境。编辑 Master 的 puppet.conf 文件，将 manifest = /etc/puppet/manifests/site.pp 改为 manifest = /etc/puppet/manifests/site.rb，修改后重新启动 Master 守护进程，这时 Agent 访问 Master 会将装载 site.pp 文件改为装载 site.rb 文件。

以下为 Ruby DSL 目录结构，我们对比一下与传统目录是否有区别（限于篇幅，笔者手动删去了大部分不相关的文件与目录）。

```
.
├── manifests
│   └── site.rb   # 站点导航
└── modules
    └── base
        └── manifests
            ├── init.rb     # 引导文件
            ├── package.rb      # 软件包安装文件
            └── create_user.rb      # 批量创建用户文件
```

从目录结构可以看出，它与传统的 Puppet 管理配置服务器方式并没有太大的区别，唯一的区别是文件扩展名由 .pp 替换成了 .rb。下面来看一下目录结构中各个文件。

1）site.rb 文件。以下为 Ruby DSL 编写的 site.rb 代码文件：

```
node /web.example.com/ do
  include base
end # node 结束符
```

以上代码文件含义：匹配 web.example.com 站点并加载 base 库文件。

2）init.rb 文件。以下为 Ruby DSL 编写的 init.rb 代码文件：

```
import create_user.rb
import package.rb
```

以上代码文件含义：加载 create_user.rb 文件与 package.rb 文件内容。

3）create_user.rb 文件。以下为 Ruby DSL 编写的 create_user.rb 代码文件：

```
hostclass :create_user do
      i = 1
      while i <= 100
            # 调用 user 资源批量创建以 user_ 为开始后接数字的账户
            user "user_#{i}", :ensure => :present
      i= i +1
      end # while 循环结束符
end # hostclass 结束符
```

通过 hostclass 声明 create_user 类，其中 hostclass 关键字等同于 Puppet 中的 class。通过 Ruby DSL 中的 while 循环调用 user 资源在 Agent 上批量创建账号，结果如下：

```
notice: /Stage[main]//User[user_1]/ensure: created
notice: /Stage[main]//User[user_2]/ensure: created
...
notice: /Stage[main]//User[user_100]/ensure: created
```

当 Agent 的 Hostname（web.example.com）访问 Master 时就会应用这些配置，在 Agent 上创建相应的账号。

案例 2 通过 Ruby DSL 与 MySQL 组合安装软件包

除了使用 Ruby 语言外，Ruby DSL 更实用的是与 MySQL 数据进行结合。以安装软件

包为例，将要安装的软件包和软件包的版本存放到 MySQL 数据库中，通过 Ruby DSL 与 MySQL 数据库结合来安装这些软件包，这也使得我们通过 Puppet 管理配置更加灵活。在安装软件包前首先需要确认 MySQL 数据库是否已经安装，如果没有安装的话可以通过以下命令来安装。

```
# gem install mysql
# gem install ruby-mysql
```

MySQL 数据库成功安装后，通过以下方式来创建数据库、表和追加数据。

```
# mysql -u root -p     # 进入 mysql 管理界面
mysql> create database cmdb;     # 创建 cmdb 库
mysql> use cmdb;    # 切换到 cmdb 库
mysql> create table packages (id VARCHAR(2), name VARCHAR(40), version
VARCHAR(20)); # 创建表
mysql> insert into packages values ( 1,"vim","2:7.2.330-1ubuntu4");    # 插入数据库
mysql> quit;
```

确认数据库安装完毕后，建立数据库名与表，并向数据库中追加数据。其中，1 表示追加数据的序号；vim 表示安装的软件名；2:7.2.330-1ubuntu4 为 vim 软件包的发行版本。

以下为 Ruby DSL 编写的 site.rb 代码文件：

```
require 'mysql'     # 加载 mysql 库文件
hostclass :packages do     # 定义 packages 类
 conn = Mysql.new('localhost', 'root', '', 'cmdb') # 连接数据库
 pkgs = conn.query('select * from packages') # 查询 packages 表内容，并将结果装入 pkgs 对象中
 # 将遍历后的数据传入 package 资源，来依次安装软件包
 pkgs.each_hash { |p| package p['name'], :ensure => p['version'] }
 conn.close
end     # hostclass 结束符
node 'default' do     # 定义默认节点
 include 'packages'     # 加载 packages 类
end     # node 结束符
```

以上代码片段含义：读取 MySQL 数据库中的软件包记录，并调用 package 资源进行安装。

10.3 Puppet 的关系图

通常情况下网站或企业内部网由多名运维工程师协作完成日常的运维工作。随着我们对 Puppet 了解和使用的深入，它的代码也变得越来越复杂，类与类和资源与资源之间的关联关系变得越来越不清晰，同时伴随着业务的增长和人员的交替，如果再不去梳理这种关联，让这种情况继续下去，就有可能引起配置逻辑错误的风险，最终影响线上的服务。目前 Puppet 提供了两种解决方案来解决日常运营工作中面临的问题。

方案 1：通过 puppt doc 工具生成文档并不断更新与沉淀。这是解决代码复杂比较好的方法之一。

方案 2：Puppet 可以与 DOT 语言结合，并通过 Graphviz 工具绘制图形的方式，画出类与资源之间的关联关系图，让我们一目了然地了解资源之间的关联关系。它与 puppet doc 结合就能很好地解决模块之前关联不清晰和逻辑复杂等一些日常运维中遇到的问题。

关于 puppet doc 工具的使用，我们曾在第 4 章中介绍过，这里不赘述。本节主要介绍 DOT 语言和 Graphviz 工具是如何帮助 Puppet 绘制关系图的。

10.3.1 DOT 语言

DOT 语言是一种文本图形描述语言，它提供了一种简单的描述图形的方法，可以直观表达一些思想或一个程序的流程，并且可以为人类和计算机程序所理解。DOT 语言以 .gv 或 .dot 为扩展名。如将 a、b、c 和 d 看成模块，以下代码片段就是通过 DOT 语言来描述模块之间的关系的。

```
# test.dot
digraph graphname {
    a -> b -> c;     # 描述模块之间的关系，a 模块下游为 b 和 c
    b -> d;     # b 模块下游为 d
}
```

如以上代码所示：DOT 语言中通过 -> 描述模块与模块之间的关系，并通过 Graphviz 工具将 DOT 语言代码转为如图 10-2 所示的有向图，通过图形的方式来解析模块之间的关系。

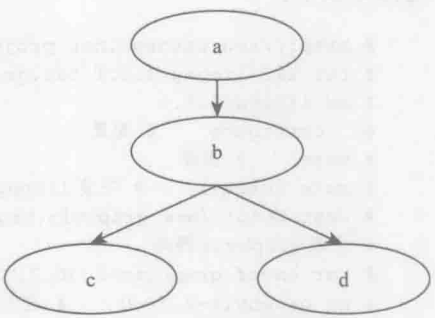


图 10-2 有向图

这只是 DOT 语言可以生成的图形之一，它还可以生成无向图，并以流程图和树形图的方式进行展示，关于 DOT 语言更多的展示方式可以参考 http://www.graphviz.org/Gallery.php 网站。如果觉得 DOT 语言的代码比较复杂，还可以增加注释使其更为详细，这有助于我们日后读懂代码逻辑。DOT 语言中支持 C 语言和 C++ 语言中的单行注释与多行注释，也支持 Shell 脚本的 # 注释方式。具体示例如下：

```
// 单行注释
 /* 多行
    注
    释 */
```

Puppet 可以生成 DOT 语言的代码，并借助 Graphviz 工具最终绘制图形。那么什么是 Graphviz 工具呢？如何安装呢？

10.3.2 Graphviz 的安装

Graphviz 是一款开源的图形可视化软件，由贝尔实验室的科学家们开发。它在网络、生物信息学、软件工程、数据库、网页设计和机器学习领域中有着广泛的应用。目前 Graphviz 支持 Fedora、RedHat、Ubuntu、Solaris、MacOS 和微软的 Windows 系列操作系统。以下为 RedHat 发行版本安装 Graphviz 的两种方式，这里笔者推荐使用 yum 安装方式。

1. yum 安装方式

通过 yum 方式来安装 Graphviz 软件包，Graphviz 中包含了 dot 命令与相关扩展包，但它默认并不支持 PNG、GIT 和 JPG 等图像格式，还需要安装 graphviz-gd 软件包。安装命令如下[⊖]：

```
# yum install graphviz graphviz-gd
```

2. 源码编译方式

通过源码编译方式安装 Graphviz 软件包。我们以生成 PNG 图像格式为例，首先需要编译安装 libpng 包，用于支持生成 PNG 图像，接着再编译安装 Graphviz 软件包。安装过程与命令如下：

```
# http://sourceforge.net/projects/libpng/     # 下载支持 libpng 的源码包地址
# tar zxf libpng-1.5.1.tar.gz     # 解压 libpng 软件包
# cd libpng-1.5.1
# ./configure     # 配置
# make     # 编译
# make install     # 安装 libpng 软件包
# wget http://www.graphviz.org/pub/graphviz/stable/SOURCES/graphviz-2.36.0.tar.gz
# 下载 graphviz 源码
# tar -xvzf graphviz-2.36.0.tar.gz     # 解压缩源码
# cd graphviz-2.36.0     # 进入解压缩的目录
# ./configure --prefix=/etc/puppet/graphviz     # 指定安装目录
# make     # 编译
# make install     # 安装
# ln -s /etc/puppet/graphviz/bin/dot /usr/bin/dot     # dot 二进制文件的软链接
```

10.3.3 Puppet 与 Graphviz 结合生成关系图

可以通过 Graphviz 工具来生成 Puppet 的关系图，首先以 Agent 安装 Nginx 服务的代码逻辑为例，看一下生成关系图的案例。Puppet 的代码如下：

```
# site.pp
# 安装 Nginx
```

⊖ 以空格做分隔，这里实际安装的是两个软件包 graphviz 和 graphviz-gd。

```
package { "Nginx":
 ensure => install,
 before => File['/usr/local/nginx/conf/nginx.conf'],
}
# 从Master同步配置文件
file { '/usr/local/nginx/conf/nginx.conf':
 mode => '600',
 source => 'puppet:///files/nginx/nginx.conf',
 notify=> Service['nginx'],
}
# 启动Nginx守护进程
service { 'Nginx':
 ensure => 'running',
 enable => true,
}
```

下面简单分析一下上面代码片段：标题为 Nginx 的 package 资源主要用来安装 Nginx 软件包，成功安装软件包后会通知标题为 /usr/local/nginx/conf/nginx.conf 的 file 资源将 Nginx 软件的 nginx.conf 配置文件由 Master 服务器同步到 Agent 的指定位置，成功同步后通知标题为 Nginx 的 service 资源启动 Nginx。由于它在每个资源中使用了资源的公有属性来建立资源与资源之间的关系，我们可以通过这一功能来生成 DOT 语言，并最终转为有向图，通过图形查看资源之间关系。

接着通过 Puppet 的 --graph 参数生成 DOT 语言代码。以下为详细命令：

```
# puppet agent --server example.com --test --graph
```

通过 graph 参数，默认情况下 Puppet 会生成 3 个 DOT 语言源文件，分别是 expanded_relationships.dot、relationships.dot 和 resources.dot，并默认存放在 /var/lib/puppet/state/graphs/ 目录中。3 个文件的作用分别如下。

- relationships.dot：显示资源之间的关系。
- resources.dot：显示资源、类之间的层次结构关系。
- expanded_relationships.dot：更详细的资源、类之间的关系。

以 resources.dot 为例。Puppet 生成的 DOT 语言代码片段如下：

```
# resources.dot
digraph Relationships {     # 图片名
   label = "Relationships"     # 图片的标题
   "File[/usr/local/nginx/conf/nginx.conf]" [     # Puppet提取到的file资源标题
      fontsize = 8,     # 生成图后的字体大小
      label = "File[/usr/local/nginx/conf/nginx.conf]"     # 生成图后的标题
   ]
   "Package[Nginx]" [     Puppet提取到的package资源标题
      fontsize = 8,     # 生成图后的字体大小
      label = "Package[Nginx]"     # 生成图后的标题
   ]
```

```
        "File[/usr/local/nginx/conf/nginx.conf]" -> "Service[nginx]" [    # 用"->"建立
资源的关系
            fontsize = 8    # 关系图中的字体大小
        ]
        "Package[Nginx]" -> "File[/usr/local/nginx/conf/nginx.conf]" [    # 用"->"建立
资源的关系
            fontsize = 8    # 关系图中的字体大小
        ]
    }
```

最后通过 Graphviz 工具将 DOT 语言转为图片。转换为图片的命令格式如下：

```
# dot -t 图片格式 DOT语言源文件路径 -o 生成后图片位置
```

目前支持 EPS、GIF、JPE、JPEG、JPG、PNG、PS 和 SVG 等图片格式。转换命令如下：

```
# 转为为 PNG 格式图片
dot -Tpng /var/lib/puppet/state/graphs/relationships.dot -o relationships.png
#  转换为 gif 格式图片
dot -Tgif /var/lib/puppet/state/graphs/resources.dot -o resources.gif
# 转换为 jpe 格式图片
dot -Tjpe /var/lib/puppet/state/graphs/expanded_relationships.dot -o expanded_
relationships.jpe
```

以 Relationships.dot 生成的图片为例。从图 10-3 中一目了然地看到 Puppet 代码片段中资源之间的关联关系。在一些复杂的代码逻辑下，通过关系图的展示可以更清晰地了解代码之间的逻辑。

当 Puppet 代码变得越来越复杂时，可以通过这种方法一目了然地了解资源间的关系。与 Puppet doc 工具结合来使用效果则更好，可以生成一份详细的运维文档，供系统维护人员参考，以便进行不定期的维护。

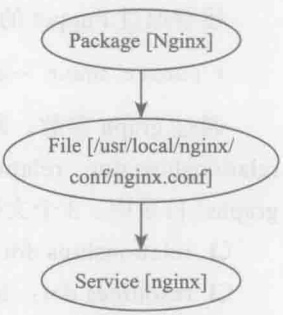


图 10-3 资源关联关系图

10.4 puppetlabs-stdlib 详述

本节主要介绍 puppetlabs-stdlib，它是 Puppet 官方提供的众多扩展工具包之一。puppetlabs-stdlib 包含了丰富的自定义函数和部分的资源、提供者与 Facter 扩展等。本节重点介绍它的函数功能，同时也为下一节的 Puppet 扩展做铺垫。如果读者未来的工作中有独立开发函数的需求，可以从本节开始先了解函数目录结构、函数作用和函数工作原理，为后续开发做铺垫。目前 puppetlabs-stdlib（下称 stdlib）共提供了 3 个版本，从表 10-1（Y 代表支持、N 代表不支持）中可以看出，它们对 Puppet 版本的支持并不同，所以使用 stdlib 时需要注意。

表 10-1 puppetlabs-stdlib 与 Puppet 版本对照表

Puppet 版本	小于 2.6	2.6	2.7	3.x
stdlib 2.x	N	Y	Y	N
stdlib 3.x	N	N	Y	Y
stdlib 4.x	N	N	Y	Y

我们可以通过 puppet module 来安装 stdlib 扩展包，操作非常简单。具体如下：

```
# puppet module install puppetlabs-stdlib
/etc/puppet/modules
└── puppetlabs-stdlib (v4.1.0)    # 包名（版本号）
```

通过以上方式安装，Puppet 会从 https://forge.puppetlabs.com 下载 stdlib 的安装包，并将它安装到 /etc/puppet/modules/stdlib 目录中。

目前 stdlib 提供了 84 个函数，笔者将它们分为了 5 类，如表 10-2 所示。后面会结合案例介绍常用的 chmop、chop、concat、unique、range、dirname、delete、delete_at、strftime 和 time 函数，其他的函数与我们介绍的常用函数使用方式类似，这里就不再一一提及，后续使用时再单独的介绍。了解 stdlib 更多的信息请参考 https://github.com/puppetlabs/puppetlabs-stdlib。

表 10-2 stdlib 函数表

分 类	函 数 名
字符串函数	abs、chomp、chop、lstrip、count、downcase、empty、rstrip、floor、getparam、getvar、max、min、pick、prefix、reject、reverse、size、sort、squeeze、strip、swapcase、unique、upcase、uriescape、capitalize
hash 函数	merge、has_key、keys
数组函数	any2array、concat、flatten、grep、join、delete、delete_at、reject、suffix、member、values、values_at、zip
验证函数	validate_absolute_path、validate_array、validate_augeas、validate_bool、validate_cmd、validate_hash、validate_re、validate_slength、validate_string、is_array、is_domain_name、is_float、is_function_available、is_hash、is_integer、is_ip_address、is_mac_address、is_numeric、is_string、has_interface_with、has_ip_address、has_ip_network
其他	Loadyaml（装载 yaml 格式）、shuffle（生成随机数）、bool2num（布尔转换数值函数）、str2bool（字符串型转换为布尔）、dirname（获取文件路径）、to_bytes（转换 bytes）、type（根据输入返回类型，包含字符串、数组、哈希、浮点型、整形和布尔型）、parsejson json（转换 Puppet 结构）、parseyaml yaml（转换 Puppet 结构）、get_module_path（获取模块路径）、fqdn_rotate、defined_with_params（判读参数是否定义）、str2saltedsha512（加密函数）、time（生成时间戳）、strftime（根据参数生成时间）、range（根据参数设置生成范围值）

（1）chmop 与 chop 函数

chmop 与 chop 函数主要用于删除“\n”与“\r\n”，其中“\r\n”表示换行加回车，“\n”表示换行。chmop 函数删除“\n”与“\r\n”后使光标定位到行首，而 chop 函数删除后使光标下移一格。以下为代码片段：

```
# chmop(hello\n)
# 返回 hello
(_) 光标所在处
# chop(hello\r\n)
# 返回 hello (_) 光标所在处
```

（2）concat 合并函数

concat 函数可以将两个数组合并为一个新数组，它包含两个参数，分别为要进行合并的两个数组。以下为代码片段：

```
# concat(['1','2','3'],['4','5','6'])
# 返回 ['1','2','3','4','5','6']
```

以下代码片段通过 concat 函数合并安装软件数组，并通过 package 资源对合并后的新数组进行安装。

```
$app_array = ['apache','mysql','memcached']
$base_array = ['mod_ssl','php','php-mysql']
$install_array = concat($app_array,$base_array)     # 合并后的数组通过 package 资源安装数组中软件包
package{ "$install_array":
 ensure => present
}
```

（3）unique 排重函数

unique 函数用于对数组或字符串进行排重。以数组 ["a","a","b","b","c","c"] 为例，经过 uniuqe 函数的处理会将重复出现的值删除，并输出去重后的结果。以下为代码片段：

```
# unique(["a","a","b","b","c","c"]) 原数组
# 返回 ["a","b","c"]     去重后结果
# unique("aabbcc") 原字符串
# 返回 abc     去重后结果
```

（4）range 根据参数生成范围值

range 函数可以根据参数来生成范围值。它包含两个参数，第一个参数为起始值，第二个参数为终止值，range 函数会根据两个参数值的来自动补充中间范围值。以下为代码片段：

```
# range("0","9")     # 生成 0 ～ 9 之间的数字
# 返回 [0,1,2,3,4,5,6,7,8,9]
# range("a", "c")     # 生成 a ～ c 之间的字母
# 返回 ["a","b","c"]
```

（5）dirname 获取文件路径函数

dirname 函数可以根据输入文件绝对路径返回文件的目录位置。以下为代码片段：

```
# dirname('/path/to/a/file.ext')
# 返回 '/path/to/a'     file.ext 文件所在路径
```

（6）delete 与 delete_at 删除函数

delete 函数用于删除数组、hash 值和字符串中的值。它包含两个参数，第一个参数为输入值（可以是字符串、数组和 hash 值），第二个参数为要删除内容。以下为代码片段：

```
# delete(['a','b','c','b'],'b')    # 删除数组中的b
# 返回 ['a','c']
# delete({'a'=>1,'b'=>2,'c'=>3},'b')     # 删除 hash 值中的b
# 返回 {'a'=>1,'c'=>3}
# delete('abracadabra','bra')     # 删除字符串中的bra
# 返回 'acada'
```

delete_at 函数用于删除输入值的位置。它包含两个参数，第一个参数为输入值，第二个参数为删除的位置。以下代码片段中 ['a','b',"c'] 为输入的数组，由于数组以下标 0 开始，所以删除第一个值就是数组中的“b”，最终返回结果为 ['a','c']。以下为代码片段：

```
# delete_at(['a','b','c'], 1)
# 返回 ['a', 'c']
```

（7）strftime 与 time 时间函数

strftime 与 time 为时间函数，主要生成规定格式的时间和 timestamp 时间戳，它们是比较常用的两个函数。

如当 Master 通过 file 资源向 Agent 同步配置文件时，如果设置了 backup 属性，Agent 会将同步的配置文件在本地目录下备份后再从 Master 同步最新配置文件。由于 Puppet 本身并不提供时间函数，我们可以借助 strftime 或 time 函数，将生成的备份文件名追加时间戳，方便我们了解备份文件的具体时间。以下代码片段为通过 strftime 或 time 函数备份文件。

```
# $suffix = time()    1396088977
# $suffix = strftime("%Y-%m-%d")    2012-12-12
file {'/etc/httpd/conf/httpd.conf':
  mode =>'777',
owner =>'httpd',
group =>'httpd',
  backup => ".$suffix.bak",
  source => "puppet: ///files/httpd/httpd.conf",
}
```

如以上代码所示，如果使用 time 函数默认会生成 httpd.conf.1396088977.bak 备份文件；如果使用 strftime 函数默认会生成 httpd.conf.20121212.bak 备份文件。可以看到 strftime 函数要比 time 函数更好识别，同时比 time 函数更加强大。strftime 可以根据设置的参数输出不同的日期格式值，如表 10-3 所示。关于 strftime 函数的更多参数请参考 https://github.com/puppetlabs/puppetlabs-stdlib 中的 strftime。

表 10-3 strftime 常用参数表

参 数	含 义	参 数	含 义
%a	周（Sun）	%A	周（Sunday）
%b	月（Jan）	%B	月（January）
%d	日（01..31）	%S	秒（00..60）
%e	日，无前导 0（0..31）	%M	分（00..59）
%I	12 小时（01..12）	%H	24 小时（00..23）
%Y	年（2014）	%m	月（01-12）

10.5 Puppet 扩展

在了解了上节的 stdlib 的工作原理、目录结构和使用方法的基础上，本节将介绍如何通过 Ruby 语言扩展独立的函数、资源与提供者功能。在这之前，首先来了解一下 Puppet 扩展程序存放的位置，在 Puppet 中有两个位置可以存放扩展程序，但是错误的位置可能会降低 Puppet 的性能，所以在开发扩展前需要了解目录的结构与不同；然后介绍 Puppet 的函数扩展；最后介绍 Puppet 的类型与提供者。

10.5.1 Puppet 扩展的目录结构

目前 Puppet 的扩展源程序可以存放在两个位置：

- 位置 1：默认存放位置位在 $rubysitedir/puppet/parser/functions/ 目录，其中 $rubysitedir 为 Ruby 安装目录位置的变量，通常 Ruby 会与 Puppet 安装位置一致。
- 位置 2：存放在 /etc/puppet/modues/custom，其中 custom 为自定义基础模块目录。

这里推荐使用位置 2 来存放扩展程序，因为位置 1 为 Puppet 程序主目录，每次 Puppet 运行时都会加载一次扩展程序，免不了会降低 Puppet 的性能。位置 2 在使用时可以通过 include 函数实现随用随调，从而不影响 Puppet 的性能。将已经开发好的文件 write_line_to_file.rb（自定义函数）、svn.rb（扩展提供者）和 repo.rb（扩展资源）存放在这两个位置的方式如下：

（1）存放在位置 1

根据 Ruby 安装的方法不同，存放的位置也不一样。可以通过 Facter 工具查看 $rubysitedir 变量找到 Ruby 和 Puppet 的安装目录。在下面的例子中笔者就是通过源码编译安装 Ruby 以及 Puppet 环境目录位置。

```
# facter | grep rubysitedir     # 查找 Puppet 库文件目录位置
rubysitedir => /usr/local/puppet/lib/ruby/site_ruby/1.8
# cd /usr/local/puppet/lib/ruby/site_ruby/1.8     # 进入库文件目录
../puppet/parser/functions/     # 扩展函数目录
../puppet/provider/custom/      # 扩展提供者目录
../puppet/parser/functions/      # 扩展类型目录
```

在存放位置1的目录结构基础上，分别创建write_line_to_file.rb、svn.rb和repo.rb源码文件。

```
# touch /usr/local/puppet/lib/ruby/site_ruby/puppet/parser/functions/write_line_
to_file.rb
# touch /usr/local/puppet/lib/ruby/site_ruby/puppet/provider/custom/svn.rb
# touch /usr/local/puppet/lib/ruby/site_ruby/puppet/parser/functions/repo.rb
```

（2）存放在位置2

自定义模块目录结构位置如下：

```
/etc/puppet/modules
└── custom
    └── manifests
    |       └── init.pp
    └── spec
    └── tests
└── lib
    └── facter
    └── puppet
        ├── parser
        |   └── functions/  # 扩展函数目录
        ├── provider/   # 扩展提供者目录
        └── type/repo.rb    # 扩展类型目录
```

- init.pp是模块必须加载的初始化文件（没有此文件，通过include函数加载时会报错）。
- functions目录是扩展函数的源文件存放目录。
- provider目录是Puppet“资源”中提供者源码文件存放目录（在本节资源又称为“类型(type)”）。
- type目录是Puppet“资源”源码存放目录。

在存放位置2的目录结构基础上，分别创建write_line_to_file.rb、svn.rb和repo.rb源码文件。

```
# touch /etc/puppet/modules/custom/lib/puppet/parser/functions/write_line_to_
file.rb
# touch /etc/puppet/modules/custom/lib/puppet/parser/provider/svn.rb
# touch /etc/puppet/modules/custom/lib/puppet/parser/type/repo.rb
```

10.5.2 Puppet函数扩展

本节主要介绍如何扩展Puppet的自定义函数。我们首先来了解扩展自定义函数过程中需要注意的事项：

- 在一个完整的Puppet C/S架构中，函数由编译器来执行。函数只能运行在Master端

或 Agent 端，两者之间的函数并不能相互的访问彼此的文件与资源。

- Master 守护进程启动前，会将现有函数装载到内存中，如果对已有的扩展函数进行了代码变更，需要及时重启 Master 的守护进程，新的函数功能才会生效。
- 函数主要在 Master 端使用，这意味着任何与函数有关的文件、库或其他调用资源都必须存放在 Master 端上，并且它们的状态（权限、用户所属）是可用的。
- 函数编写时的源码文件名必须与源码文件内函数名一致，否则将无法装载函数。
- 通常使用 lookupvar('FACT NAME') 来代替 Facter['FACT NAME'].value 获取 Fact 值。如果 lookupvar 函数获取不到 Fact 值，在 Puppet 3 以上版本会返回 nil 值，在 Puppet 2.7 版本则返回 undefined 值。
- 扩展函数的扩展名必须是 .rb。

下面来介绍两个关于扩展函数的案例。

案例 1

通过 write_line_to_file 自定义扩展函数向文件追加内容。

这里以自定义模块目录为例，来介绍如何扩展自定义函数。write_line_to_file 函数包含两个参数，参数 1 为追加文件的目标位置，参数 2 为写文件的内容。编辑 write_line_to_file.rb，将以下内容追加到文件中。

```
    # /etc/puppet/modules/custom/lib/puppet/parser/functions/write_line_to_file.rb 自定义函数文件全路径
    module Puppet::Parser::Functions
        newfunction(:write_line_to_file) do |args| # write_line_to_file 为函数名需要与源码文件名一致
          filename = args[0]     # 参数 1
          str = args[1]     # 参数 2
          File.open(filename, 'a') {|fd| fd.puts str }
        end
    end
```

如以上代码所示，通过调用 Puppet::Parser::Functions 的 newfunction 方法传入自定义的函数名 write_line_to_file。通过 args[0] 传入写文件的目标位置；通过 args[1] 传入追加文件的内容。最后通过 File.open 方法将 args[1] 的内容追加到 args[0] 的目标文件中，这整个的流程就是 write_line_to_file 自定义函数的工作原理。在测试使用 write_line_to_file 自定义函数前还需要完成以下的步骤：

1）Puppet 在加载扩展函数过程中需要读取模块中的 init.pp 文件。在使用前可以通过追加空类的形式来建立 init.pp 文件，追加空类的原因主要是避免 Puppet 加载自定义函数时报错。以下为 custom 模块的代码：

```
    # /etc/puppet/manifests/etc/puppet/module/custom/manifests/init.pp
    class custom{
        # 内空
    }
```

2）通过编辑 write_line_to_file.pp 文件来测试我们开发的扩展函数结果。将开发的扩展函数追加到文件中。

```
# /etc/puppet/manifests/etc/puppet/manifests/write_line_to_file.pp
include custom     # 加载自定义模块
write_line_to_file('/tmp/some_file', "Hello world!")
```

由于这里我们将文件存放在了上一节提到的位置 2 中，所以需要先通过 include 函数加载 custom 模块，然后在 write_line_to_file 函数中追加文件的路径与文件名 /tmp/some_file；“Hello world”为向文件追加的内容。

3）通过 puppet apply write_line_to_file.pp 的方式来验证自定义函数是否生成了 /tmp/some_file 文件和内容。

```
# puppet apply write_line_to_file.pp
# cat /tmp/some_file
Hello world!
```

案例 2

在自定义函数中调用 facts 值。

在案例 1 的目录结构基础上，再来看如何在自定义函数中调用 facts 值。通常情况下，当 Agent 比较多而又频繁访问 Master 时，如何均衡利用 Master 资源就成为了问题。在之前的章节我们也曾讨论过这样的问题，这里仍然以它为例，通过自定义函数方式来解决这一问题。这里的解决方案其实有很多种，如通过随机数方式、IP 取模方式和 MD5 哈希 Hostname 方式，但其原理一样，均是通过 cron 资源设置 crontab 的 minute 值，使其通过随机数方式来让 Agent 随机设定时间访问 Master，达到均衡 Master 负载的目的。

通过下面的例子，我们来了解如何获取 Agent 的 IP 值，并根据 IP 值取模方式来达到均衡负载的目的。编辑 minute_from_address.rb 自定义函数，将以下内容追加到文件中。

```
# /etc/puppet/modules/custom/lib/puppet/parser/functions/minute_from_address.rb
# 自定义函数文件全路径
require 'ipaddr'
module Puppet::Parser::Functions
    newfunction(:minute_from_address, :type => :rvalue) do |args|
     IPAddr.new(lookupvar('ipaddress')).to_i % 60     # lookupvar 函数获取外部变量
    end
end
```

如以上代码所示，首先通过 require 加载 Facter 的 ipaddr 库，此库用途为获取 Agent 的 IP 值；接着通过调用 Parser::Functions 的 newfunction 方法创建 minute_from_address 函数，并通过 lookupvar 获取外部变量的值，然后进行模除；最终的结果通过函数返回。

编辑 run_some_job_at_a_random_time.pp 文件测试最终的结果。追加以下内容到文件中。

```
# /etc/puppet/manifests/etc/puppet/manifests/run_some_job_at_a_random_time.pp
include custom
cron { run_some_job_at_a_random_time:
  command => "/usr/local/sbin/some_job",
  minute => minute_from_address(),    # minute_from_address 根据 Agent 的 IP 计算随机数
}
```

当 Agent 访问 Master 时 minute_from_address 函数就会根据 Agent 的 IP 算出它的分钟值，由于不同的 IP 是不一致的，所以算出的最终分钟值也是不一致的，从而达到了 Agent 分时访问 Master 的目的。

10.5.3 Puppet 类型与提供者

第 7 章中笔者曾介绍过 Puppet 的资源，Puppet 中每一种资源用来管理配置服务器的一类功能，如创建账号资源、创建文件资源和安装软件资源等。同时 Puppet 资源的优势是管理配置工作可以不区分操作系统的发行版本，这也是因为每个资源都有自己相对独立的提供者。以 package 资源为例，它包含了 YUM、APT、GEM 和 MSI 等 20 多个提供者，而正是因为这些提供者使得 Puppet 在管理配置服务器时不区分操作系统发行版本。截至本书出版前，Puppet 官方网站共提供了 48 个资源，如果这些资源不能满足我们需求怎么办？就可以通过 type（类型，下称“类型”）与 provider（提供者）来扩展 Puppet 的新功能，这里的所说的“类型”就是我们 Puppet 的资源，而提供者就是扩展平台配置的一个接口。Puppet 的类型与提供者均用 Ruby 语言开发，在学习扩展 Puppet 的类型与提供者前最好先阅读一下 Puppet 本身自带的类型源码文件，推荐从比较简单的 user、host 和 package 资源逐渐学习到最后比较复杂的 file 资源，这样会为后续开发扩展打下良好的基础。下面通过案例来看如何开发 Puppet 的类型与提供者。

版本控制工具是运维工程师与开发工程师的必备工具，对于开发工程师来说可以通过版本控制工具并行开发，并通过版本控制工具规避并行开发带来的风险，提高开发的效率与质量。对于运维工程师来说可以对线上配置文件进行版本控制，如果在上线过程中由于配置文件的原因导致问题产生，通过版本控制工具能够及时回滚配置文件到某个历史正确的版本上，将风险降到最低，这些都是版本控制工具的优势。那么是否可以将版本控制工具与 Puppet 语言结合呢？当然是可以的。以 SVN 版本控制工具为例，可以通过 Puppet 语言来实现 Check out 功能，也就是将 SVN 中的文件或目录，通过 SVN 软件下载到本机的指定目录中。如以下的 Puppet 代码：

```
exec { "svn co http://svn.example.com/trunk/ /var/www/wp":
    creates => "/var/www/wp",
}
```

如以上代码所示，Puppet 通过 exec 资源调用 SVN 的命令，将 http://svn.example.com/trunk/ 地址中的目录 Check out 到本机的 /var/www/wp 目录。其中 SVN 命令中的 co 是

Check out 的简写形式，其含义是导出 SVN 中的数据到指定位置。这种方式很不错，但有没有更好的方式呢？其实更好的方式是将整个 SVN 的 Check out 功能封装成 Puppet 的类型，并通过资源的形式来导出数据，这种方式也是本节将重点介绍的。

下面我们来扩展一个版本控制工具的类型，用于管理我们的版本配置仓库。扩展的类型名为 repo，在介绍扩展前需要了解以下 3 点：

- 类型的源文件存放位置为 /etc/puppet/modules/custom/lib/puppet/type/ 目录。
- 类型的扩展文件需要与声明类型名一致，并且扩展文件需以 *.rb 结尾。
- 如果在 Master/Agent 上都需要使用我们扩展的类型与提供者，则需要开启 Master/Agent 两端的 puppet.conf 配置文件中的 pluginsync=true 功能，通过此功能 Master 就会将开发后的类型源文件同步到 Agent 使用（需要注意的是，如果使用的是 Puppet 2.7 或更低版本，pluginsync 默认为 false（关闭），需要将它手动设置为 true（打开），并重启守护进程。如果使用的是 Puppet 3 版本，pluginsync 默认为 true）。

下面开始扩展 Puppet 的类型。编辑 repo.rb 将以下内容追加到文件中。

```
# /etc/puppet/modules/custom/lib/puppet/type/repo.rb
Puppet::Type.newtype(:repo) do     # 声明资源，注意声明的 repo 资源名与文件名需要一致
   @doc = "Manage repos"     # 指定文档
   ensurable     # 开启 ensure 属性的快捷方式
   # 资源的参数
   newparam(:source) do
      desc "The repo source"
      validate do |value|     # validate 钩子函数用于遍历 source 值
         if value =~ /^git/     # 匹配 git
            resource[:provider] = :git     # 将 git 加入提供者数组
         else
               resource[:provider] = :svn     # 否则将 svn 加入提供者数据组
            end
      end
      isnamevar     # 使用参数作为标题
end
# 资源的参数
newparam(:path) do
   desc "Destination path"
   validate do |value|
      unless value =~ /^\/[a-z0-9]+/  # 判断不匹配 "^\/[a-z0-9]+" 则抛出错误信息给 Puppet
               raise ArgumentError , "%s is not a valid file path" % value
      end
   end
  end
end
```

如以上代码所示，Puppet 通过 Puppet::Type 中的 newtype 方法来创建类型，类型名为 repo，需要与文件名（repo.rb）一致；@doc =“Manage repos”指定类型的文档；ensurable 用来创建 ensure 属性的快捷方式；newparam 方法用来创建类型的属性，如 source 属性指定

版本控制工具的源；path 属性指定版本控制工具的输出目标；validate 为钩子函数，用于输入值的合法性判断。

对 validate 钩子函数举例如下：

```
newparam(:source) do
      validate do |value|     # 钩子函数，用于输入值的合法性判断
         if value =~ /^git/
            resource[:provider] = :git
         else
            resource[:provider] = :svn
         end
      end
end
```

validate 钩子函数会匹配 source 属性的值，如果匹配到了 git 则将 git 参数赋值到 resource 数组中的 provider，如果匹配到了 SVN 则将 SVN 参数赋值到 resource 数组中的 provider。resource 数组中的 provider 用于最终确定使用哪种提供者；isnamevar 使用参数作为类型的标题；最后通过 raise ArgumentError 向 Puppet 抛出错误信息。

一个简单的 repo 类型就开发好了，但它还不能正常的工作，还需要提供者的配合。这里以 SVN 的提供者为例，创建以下目录。

```
/etc/puppet/modules/custom/lib/puppet/provider/repo/    # 在 provider 目录下创建与类型一致的 repo 目录
```

在 ../provder/repo 目录中编辑 svn.rb 文件，并追加以下内容到文件中。

```
# /etc/puppet/modules/custom/lib/puppet/provider/repo/svn.rb
require 'fileutils'     # 加载工具库
Puppet::Type.type(:repo).provide(:svn) do     # 声明 repo 资源的 svn 提供者
   desc "SVN Support"
   commands :svncmd => "svn"
   def create     # 创建资源命令
      svncmd "checkout", resource[:name], resource[:path]
   end
   def destroy     # 销毁资源命令
      FileUtils.rm_rf resource[:path]
   end
   def exists?     # 判断资源命令
      File.directory? resource[:path]
   end
end
```

如以上代码所示，repo 类型的提供者主要围绕着 3 个方法，它们分别是：

- create（创建资源方法）
- destory（销毁资源方法）
- exists（判断资源方法）

首先代码加载了 fileutils 库，因为在销毁方法中使用到了 fileutils 库中的删除方法；接着通过 Puppet::Type.type(:repo).provide(:svn) 创建 repo 类型的 SVN 提供者；然后定义提供者的命令 commands :svncmd => “svn”；最后在 3 个方法中分别作处理。

（1）create 创建资源方法

svncmd 方法会将 SVN 原路径中的数据 checkout 到目标路径中。

```
def create
  # 如果将以下命令转换为 SVN 命令，它等于 svn checkout http://svn.example.com/trunk /var/www/wp
svncmd "checkout", resource[:name], resource[:path]
end
```

（2）destroy 销毁资源方法

Puppet 会将目标路径中已经存在的数据删除。

```
def destroy
 # 如果将以下命令转换为系统命令，它等于 rm -rf /var/www/wp
FileUtils.rm_rf resource[:path]
end
```

（3）exists 判断资源方法

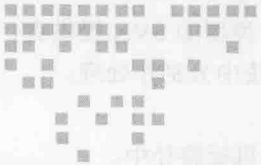
exists 方法会检查目标数据是否存在，此方法的状态通常与 ensure 属性的值联系紧密。如果 exists 方法返回为 false 并且 ensure 被设置为 present，那么 create 方法就会被调用；如果 exists 方法返回为 true 同时 ensure 被设置为 absent，那么 destory 方法就会被调用。

```
def exists?
 File.directory? resource[:path]
end
```

我们成功添加 svn.rb 提供者后，就可以通过以下方式来使用创建好的 repo 资源了。编辑 repo.pp 将以下代码追加到文件中。

```
include custom
repo { "svn_test":
   source => "http://svn.example.com/trunk",     # SVN 的源路径
   path => "/var/www/wp",     # Check out 的目标路径
   ensure => present,
}
```

当 Agent 访问 Master 执行到此资源时就会从 SVN 中 check out trunk 目录到 Agent 的 /var/www/wp 目录，其最终执行结果与 svn co http://svn.example.com/trunk/ /var/www/wp 一致。



Chapter 11 第 11 章

Puppet 集群技术

架设好 Puppet 的 C/S 结构后，随着业务的增长可能会经常遇到部分 Agent 的配置文件与 Master 同步后不一致的状况，进而慢慢导致 Master 负载不断变高、下发配置的时间变长，类似的不稳定问题接踵而至，这些问题很有可能是由 Master 达到了服务的处理瓶颈所导致的。有的读者不禁会问，官方网站提出用最低配置的 Puppet 服务器可以管理 1000 个节点的配置与访问，而我的 Master 服务器配置比较高，又小于 1000 个节点为什么也会出现类似情况呢？在这里告诉大家，这是由于 Puppet 所承载的业务形态不同，处理能力也截然不同。本章将介绍如何避开 Puppet 的短板，从而提高 Master 的处理性能。如 Puppet 本身并不适合大文件的传输，而在使用过程中常常需要通过 Master 向 Agent 同步较大的数据文件，在几百台 Agent 同时访问 Master 时就会达到 Master 的处理瓶颈，影响正常的 Agent 配置同步，这时就可以通过 Puppet 的 Exec 资源调用 Rysnc 来同步配置文件绕开 Puppet 的短板。类似的情况有很多，这些都可在本章找到答案。

本章首先从 Agent 到 Master 端由浅入深地介绍 Master 处理瓶颈的解决方案，读者可根据自己的情况来选择适合自己的方案。建议读者能使用单机方式提高 Puppet 性能就不要使用集群来管理，尽量让架构变得简单可维护；接着通过 LVS 和 DNS 来介绍建立 Puppet 集群技术方案；最后介绍 Puppet 认证瓶颈的解决方案。

11.1 Master 单机瓶颈解决方案

本节主要介绍在单机情况下，如何通过 Master 的配置来解决处理瓶颈。Master 默认使用 WEBRick 提供服务，它是一款单进程的 Web 服务器，处理能力有限，但是可以启动多

Master 守护进程，通过多进程的方式提升它的处理性能。Master 可以通过参数方式启动多进程提供服务，在 Agent 连接 Master 时通过脚本将整个访问逻辑封装起来，通过脚本生成随机数，以随机的方式访问 Master，从而降低 Puppet 单进程处理压力，提高 Master 处理性能。以下为 Master 与 Agent 的封装脚本。

1. Master 端

Master 多进程启动脚本。编辑 master_multiport.sh，将以下内容追加到脚本文件中。

```
#!/bin/bash
# author: wds
# time : 20140218
# 设置默环境变量
PATH="/sbin:/usr/sbin:/bin:/usr/bin:/usr/local/bin/"
export PATH
export LC_ALL=POSIX
export LANG=en
# 设置 Master 启动端口的数组
array=(8140 8141 8142 8143)
for i in ${array[@]}
do
# 启动 Master 端口
nohup puppet master --no-daemonize --verbose --masterport  $i --pidfile=/tmp/puppet_"${i}".pid 2>&1 >> /tmp/puppet.log &
done
```

如以上代码所示，为了避免脚本运行过程中出现命令找不到或者乱码等问题，首先设置脚本的环境变量。接着声明要启动的端口数组，将启动端口放入 array 数组中，并通过 for 循环来遍历此数组，在 for 语句遍历过程中通过 puppet 命令启动 Master 的守护进程。启动参数如下。

- no-daemonize 参数：将进程输出信息发送到标准输出。
- verbose 参数：输出扩展信息。
- masterport 参数：自定义 Master 的端口。
- pidfile 参数：指定端口的 pid 文件。

脚本中的 for 语句会通过 puppet 命令依次启动 array 数组中的端口。最后通过 2>&1 >> /tmp/puppet.log 将启动过程追加到 puppet.log 文件中。可以通过 netstat --tnl 命令确认最终的执行结果。如果端口启动失败，可以通过 /tmp/puppet.log 启动过程日志来定位失败的原因。

```
# netstat -tnl | grep 814*
tcp        0      0 0.0.0.0:8140            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8141            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8142            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8143            0.0.0.0:*               LISTEN
```

2. Agent 端

Agent 随机访问 Master 脚本。将以下内容追加到脚本文件中，并通过封装后的 puppet.sh 脚本加入 crontab 中来访问 Master。

```
#!/bin/bash
# author: wds
# time : 20140218
# 设置默环境变量
PATH="/sbin:/usr/sbin:/bin:/usr/bin:/usr/local/bin/"
export PATH
export LC_ALL=POSIX
export LANG=en
# 初始化 Puppet 相关环境
_PWD="/etc/puppet"
_PUPPET="/usr/local/puppet/sbin/puppet agent"
_HOST="puppet.example.com"
_LOG_TIME=`date +%Y%m%d%H%M`
[ ! -e ${_PWD}/lock ] && mkdir ${_PWD}/lock
[ ! -e ${_PWD}/log ] && mkdir ${_PWD}/log
[ -e /etc/puppet/log/ ] && find /etc/puppet/log/ -name "*.log" -mtime +3 -exec rm
-rf {} \;
[ -e /etc/puppet/lock/ ] && find /etc/puppet/lock/ -name "*.lock" -mmin +40 -exec
rm -rf {} \;
# 写日志函数
writelog()
{
echo "$1" >> ${_PWD}/log/puppet.log
}
# 加载 Puppet 的环境
environment=$1
if [ "${environment}" == "test" ];then
_ENVIROMENT="testing"
elif [ "${environment}" == "development" ];then
_ENVIROMENT="development"
else
_ENVIROMENT="production"
fi
if [ -e ${_PWD}/lock/puppet.lock ];then
exit;
fi
touch ${_PWD}/lock/puppet.lock
# 随机访问 Master 策略
port=(8140 8141 8142 8143)
rand=$(($RANDOM%${#port[@]}))
if [ "x${port[$rand]}" != "x" ];then
$_PUPPET --server $_HOST:${port[$rand]}  --environment=${_ENVIROMENT} --test |
tee >> ${_PWD}/log/puppet.log
writelog "$_HOST:${port[$rand]} succ"
fi
```

```
    rm -rf ${_PWD}/lock/puppet.lock
    exit 0;
```

如以上代码所示，主要工作原理是通过在 Agent 生成的随机端口，随机访问 Master 的端口数组，以达到分进程处理的目的。整个脚本分为以下 5 个部分。

1）设置脚本的环境变量与字符集：众所周知，crontab 定时任务中不包含系统的环境变量，这样设置的主要目的是避免出现脚本运行过程中命令找不到或者乱码的问题。

2）初始化 Puppet 相关环境：创建 Puppet 运行过程中所需要用到的目录。如果已经创建目录，则根据目录中的日志文件已经创建超过 3 天的进行清零。lock 文件是运行过程中的锁文件，目的是防止脚本同时重复运行，当此文件生成超过 40 分钟，则将其删除。

3）写日志函数：用户记录 Puppet 执行过程中输出的日志等信息，方便后续查找问题。

4）加载 Puppet 环境：通过脚本的位置参数 $1 进行参数的合法性验证，确保输入参数只有 testing（测试环境）、development（开发环境）和 production（线上环境）。

5）Agent 随机访问 Master 策略：首先定义随机访问 Master 的端口数组 port，并赋值指定的 Master 端口为 (8140 8141 8142 8143)，通过 $(($RANDOM%${#port[@]})) 中的 random 函数生成随机数，${#port[@]} 值是为数组的长度，随机数模除（%）数组长度，得到一个随机的数组下标值，random 函数会随机生成 0~3 之间的数（数组下标由 0 开始），最终通过 ${port[$rand]} 变量将 random 生成的随机数下标取出。如 $rand 为 3，就会从 port 数组中取出下标为 3 的值，也就是 8143 端口。Agent 脚本会根据生成的此随机端口来访问 Master。

11.2 Mongrel 模式

本节主要介绍如何通过 Mongrel 的方式来提升 Master 的处理性能。那么什么是 Mongrel 呢？简单来说它是一款快速的、基于 Ruby 的 HTTP1.1 Web 应用服务器，可以不借助任何框架来独立运行 Ruby 开发的 Web 应用，同时也可以实现前端 Web 代理功能，并可以接收并处理 SSL 连接。但可惜的是，Puppet 3 中已经取消了对 Mongrel 的支持，所以此方式只适合在 Puppet 2.7 以下版本使用。

通常我们将 Mongrel 与 Apache 或 Nginx 的反向代理功能结合使用，提升 Puppet 并发处理性能。众所周知 Nginx 的并发处理性能要优于 Apache，所以本节主要介绍 Nginx+Mongrel 的模式，如果读者想同时了解 Apache+Mongrel 的模式，可以参考 http://projects.puppetlabs.com/projects/puppet/wiki/Using_Mongrel。

1. Mongrel 与 Nginx 安装

Mongrel 目前可以与 Puppet 2.7 以下的版本结合使用，但需要注意的是，Puppet 0.23.1 版本对 Mongrel 只做了适当的支持，因为这一版本不支持 HTTP 的 X-Client-Verify 验证头，

所以不能用作证书的签名。另外建议使用 Mongrel 1.1.5 或者更高的版本。本节以 CentOS 6.5_x86_64 位系统为例，通过 yum 方式来安装 Mongrel 和 Nginx。

```
# yum install rubygem-mongrel nginx
```

2. Mongrel 配置

安装好 Mongrel 后，修改配置文件让 Master 的守护进程以 Mongrel 方式启动。Puppet 2.7 以下版本，Master 的守护进程支持两种启动方式，默认启动方式为 WEBRick 和 Mongrel 可选方式，编辑 /etc/init.d/puppetmaster（下称 puppetmaster）启动脚本，将以下内容追加到 puppetmaster 脚本的头部。

```
PUPPETMASTER_PORTS=(18140 18141 18142 18143)
PUPPETMASTER_EXTRA_OPTS="--servertype=mongrel  --ssl_client_header=HTTP_X_SSL_
SUBJECT"
```

如以上配置所示，PUPPETMASTER_PORTS 为数组，数组中包含了启动 Master 守护进程的端口号，PUPPETMASTER_EXTRA_OPTS 为启动守护进程的扩展参数，参数中设置启动 servertype 方式为 mongrel，并设置 ssl_client_header 为 HTTP_X_SSL_SUBJECT 用于 SSL 验证。追加后可以通过以下方式启动。

```
# service puppetmaster start
Starting puppetmaster:
Port: 18140                                                [ OK ]
Port: 18141                                                [ OK ]
Port: 18143                                                [ OK ]
Port: 18144                                                [ OK ]
```

启动后别忘记通过 netstat --tnl 命令再次确认端口是否已经启动。

```
# netstat -tnl | grep 1814*
tcp        0      0 0.0.0.0:18140           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:18141           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:18142           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:18143           0.0.0.0:*               LISTEN
```

Mongrel 配置成功后 Master 暂时还提供不了服务，需要配置并启动 Nginx，将 Master 服务器的默认 8140 端口绑定到 Nginx 服务器上，并配置 Nginx 对 8140 端口转发，将请求转发到 Mongrel 的端口上来提供服务。

3. Nginx 配置

再来看 Nginx 反向代理功能的配置文件。这里以通过 yum 安装 Nginx 为例。

```
# 由于 CentOS 版本默认源中并不包含 Nginx，需要通过以下方式配置 Nginx 源
# http://nginx.org/packages/rhel/6/noarch/RPMS/nginx-release-rhel-6-0.el6.ngx.
noarch.rpm
```

```
# 安装 Nginx
# rpm -ivh nginx
```

Nginx 安装好后，接着编辑 /usr/local/nginx/conf/nginx_puppet.conf（下称 nginx_puppet.conf）文件，将以下内容追加到 nginx_puppet.conf 文件中。

```
# /usr/local/nginx/conf/nginx_puppet.conf
user    daemon daemon;
worker_processes    4;
worker_rlimit_nofile 65535;
error_log           /var/log/nginx-puppet.log notice;
pid                 /var/run/nginx-puppet.pid;
events {
      use                   epoll;
      worker_connections    32768;
}
http {
   sendfile                 on;
   tcp_nopush               on;
   keepalive_timeout        500;
   tcp_nodelay              on;
# 设置转发的端口
upstream puppetmaster {
       server 127.0.0.1:18140;
       server 127.0.0.1:18141;
       server 127.0.0.1:18142;
       server 127.0.0.1:18143;
}
# 设置 Puppet 默认 8140 端口配置
server {
      listen 8140;
      root                          /etc/puppet;
# 开启 SSL 验证功能
      ssl                           on;
      ssl_session_timeout           7m;
      ssl_certificate               /opt/puppet/ssl/certs/puppet.example.com.pem;
      ssl_certificate_key           /opt/puppet/ssl/private_keys/puppet.example.
com.pem;
      ssl_client_certificate        /opt/puppet/ssl/ca/ca_crt.pem;
      ssl_crl                       /opt/puppet/ssl/ca/ca_crl.pem;
      ssl_verify_client             optional;
# 设置匹配文件转发规则
      location /production/file_content/files/ {
            types { }
            default_type application/x-raw;
            alias /etc/puppet/manifests/files/;
      }
# 设置匹配模块转发规则
      location ~ /production/file_content/modules/.+/ {
            root /etc/puppet/modules;
```

```
            types { }
            default_type application/x-raw;
            rewrite ^/production/file_content/modules/([^/]+)/(.+)$ /$1/files/$2
break;
        }
    # 设置默认转发请求
        location / {
        proxy_pass              http://puppetmaster;
        proxy_redirect          off;
        proxy_set_header        Host              $host;
        proxy_set_header        X-Real-IP         $remote_addr;
        proxy_set_header        X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_set_header        X-Client-Verify   $ssl_client_verify;
        proxy_set_header        X-SSL-Subject     $ssl_client_s_dn;
        proxy_set_header        X-SSL-Issuer      $ssl_client_i_dn;
        proxy_buffer_size                 16k;
        proxy_buffers                     8 32k;
        proxy_busy_buffers_size           64k;
        proxy_temp_file_write_size        64k;
        proxy_read_timeout                65;
        }
      }#server end
    }#http end
```

如以上配置文件所示，整个配置文件中包含了以下 6 个部分。

1）设置转发的端口：Puppet 守护进程将端口绑定在 18140 ～ 18143 之间。通过 Nginx 的配置将默认 Puppet 的 8140 端口接收请求随机转发到 18140 ～ 18143 端口之间。

2）设置 Puppet 默认 8140 端口配置：通过 server 对 Nginx 的 8140 端口设置转发规则。规则中包含对 SSL 验证的转发、fileserver 请求转发和 modules 基础模块的转发。

3）开启 SSL 验证功能：开启 SSL 验证功能，并指定 Puppet 证书文件路径，用于处理 CA 验证请求。

4）设置匹配文件转发规则：通过 Nginx 中的 location 来匹配 /production/file_content/files/ 路径，并将请求转发到 /etc/puppet/manifests/files/ 目录中。

5）设置匹配模块转发规则：通过 Nginx 中的 location 来匹配 /production/file_content/modules/.+/ 路径，并将请求转发到 /etc/puppet/modules 目录中。

6）设置默认转发请求：将以上没有匹配到的请求路径转发到 puppetmaster，也就是我们第一部分 Puppet 守护进程绑定的端口上。

最后在配置好 Nginx 后，通过以下方式启动 Nginx：

```
# /usr/local/nginx/sbin/nginx -t /usr/local/nginx/conf/nginx_puppet.conf
```

通过 Nginx 的 -t 参数加载指定自定义的 Nginx 配置文件。可以通过 netstat –tnl | grep 8140 来查看 Nginx 端口是否已经启动。如果已经启动可以到 Agent 上通过以下方式确认测试是否可以访问。

```
# puppet agent --server puppet.example.com  --test --noop
```

最后通过 tcpdump 或查看 Nginx 的日志方式确认 Nginx+Mongrel 模式是否已经正常工作。

11.3 Phusion Passenger

Phusion Passenger（官方网站 https://www.phusionpassenger.com/）模块也被称为 mod_rails、mod_passenger 或者简称为 Passenger。Passenger 是一个用于 Ruby 程序嵌入执行的 Apache 模块，由于它并不是 Apache 的标准模块，所以需要独立的安装，这与 Apache 中的 mod_php、mod_perl 模块嵌入 PHP 和 Perl 程序非常相似。Passenger 可以工作在任何 POSIX 标准的操作系统。换句话说：现有的操作系统中，除了 Microsoft Windows 以外都可以很好地支持 Passenger，包括对 32 位和 64 位平台的支持。Puppet 官方推荐使用 Apache + Passenger 的方式来提升 Master 的处理性能。首先 Apache+Pssenger 功能强大且配置简单，支持目前 Puppet 现有版本（Puppet 2.6、Puppet 2.7 和 Puppet 3.*）；其次 Puppet 3 中已经去掉了对 Mongrel 的支持，所以建议刚接触 Puppet 的读者优先使用 Passenger 的方式来提升 Master 的处理瓶颈；最后 Passenger 除了支持 Apache 外还支持 Nginx。下面就从 Passenger 的安装讲起，首先介绍 Apache+Passenger 的环境安装、Apache 配置和启动 Apache+Passenger，接着介绍 Nginx+Passenger 的安装。Nginx 的并发处理能力要比 Apache 强，随着 Nginx 软件功能的不断完善，它受到很多企业的青睐，在 Web 软件使用排行中有逐年上升的趋势，所以这里有必要了解一下 Nginx+Passenger 的整合与使用。

11.3.1 Apache + Passenger

首先来安装 Apache+Passenger 所需要的环境。以 CentOS 6.5_64 位系统为例，整个配置流程共分为以下七步：

1）安装 Apache 与 Ruby 环境。

```
# yum install httpd httpd-devel ruby-devel rubygems mod_ssl
```

2）安装 Passenger 环境。这里推荐使用配置 gem 的淘宝源来安装 Passenger，关于淘宝源的配置方法请参考第 3 章。

```
# gem install rack passenger
```

3）安装 Passenger 后，通过 passenger-install-apache2-module 命令来自动生成 Passenger 的 Apache 扩展，但在生成扩展前 Passenger 会检查以下扩展包是否已经安装。为了避免安装过程中报错，可以提前安装并确认以下安装包。

```
# yum install gcc gcc-c++ curl-devel openssl-devel zlib-devel
```

接着运行 passenger-install-apache2-module 命令：

```
# passenger-install-apache2-module
Welcome to the Phusion Passenger Apache 2 module installer, v4.0.41.
This installer will guide you through the entire installation process. It
shouldn't take more than 3 minutes in total.
```

由于输出的内容比较多，这里做了截取。运行 passenger-install-apache2-module 命令后，按回车，最终它会输出以下配置信息。在配置 Apache 时需要将以下的配置追加到 httpd.conf 文件的尾部，所以需要先将以下信息记录到文本文件里备用。

```
<IfModule>
LoadModule passenger_module /usr/lib/ruby/gems/1.8/gems/passenger-4.0.42/
buildout/apache2/mod_passenger.so
    <IfModule mod_passenger.c>
     PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-4.0.42
     PassengerDefaultRuby /usr/bin/ruby
</IfModule>
```

4）创建 Rack 配置目录与配置。

```
# 创建 Rack 目录
# mkdir -p /etc/puppet/rack{public,tmp}
# 复制 Rack 配置文件 config.ru
# cp /usr/share/puppet/ext/rack/files/config.ru /etc/puppet/rack/
# 设置目录的权限
# chownn -R puppet:puppet /etc/puppet/rack/
```

5）编辑 Apache 的配置文件。为了避免 httpd.conf 被破坏，首先将原始 httpd.conf 文件进行备份，然后将 httpd_conf 改为 httpd_puppet.conf。具体如下：

```
# cd  /etc/httpd/  && cp httpd.conf  httpd.conf.bak && mv httpd.conf.bak  httpd_
puppet.conf
```

编辑 /etc/httpd/conf/httpd_puppet.conf 配置文件。注释掉默认的 Listen 80，并将以下内容追加到配置文件中（编辑 httpd_puppet.conf 配置文件后，可以通过 ./apachectl configtest 来验证 httpd_puppet.conf 语法是否正确）。

```
# 开始
# 加载 passenger.so 模块
LoadModule passenger_module /usr/lib/ruby/gems/1.8/gems/passenger-4.0.42/
buildout/apache2/mod_passenger.so
<IfModule mod_passenger.c>
    PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-4.0.42
    PassengerDefaultRuby /usr/bin/ruby
</IfModule>
# 配置 Puppet 虚拟主机环境
Listen 8140
<VirtualHost *:8140>
```

```
        # 开启 Apache 的 SSL 引擎
        SSLEngine on
        SSLCipherSuite SSLv2:-LOW:-EXPORT:RC4+RSA
        SSLCertificateFile      /var/lib/puppet/ssl/certs/puppet.example.com.pem
        SSLCertificateKeyFile    /var/lib/puppet/ssl/private_keys/puppet.example.com.
pem
        SSLCertificateChainFile /var/lib/puppet/ssl/ca/ca_crt.pem
        SSLCACertificateFile    /var/lib/puppet/ssl/ca/ca_crt.pem
        SSLCARevocationFile    /var/lib/puppet/ssl/ca/ca_crl.pem
        SSLVerifyClient optional
        SSLVerifyDepth  1
        SSLOptions +StdEnvVars
        # 设置访问的头文件
        RequestHeader set X-SSL-Subject %{SSL_CLIENT_S_DN}e
        RequestHeader set X-Client-DN %{SSL_CLIENT_S_DN}e
        RequestHeader set X-Client-Verify %{SSL_CLIENT_VERIFY}e
        # 指定 Rack 配置的目录，并设置目录的访问权限
        DocumentRoot /etc/puppet/rack/public/
        <Directory /etc/puppet/rack/>
           Options None
           AllowOverride None
           Order allow,deny
           allow from all
        </Directory>
    </VirtualHost>
    # 结束
```

6）httpd_puppet.conf 文件配置后，就可以启动 Apache 的守护进程提供服务了。在启动前需要先确认旧的 puppetmaster 守护进程是否已经关闭，如果未关闭系统则会报端口冲突，导致 Apache 启动失败。关闭 puppetmaster 守护进程，然后再启动 Apache 守护进程，最后通过 netstat –tnl 命令确认 Apache 是否成功启动。步骤如下：

```
    # service puppetmaster stop
    # servicd httpd start
    # netstat -tnl | grep 8140
```

7）Master 的守护进程启动后，就可以从 Agent 访问 Master 的测试结果了。笔者通过 for 语句循环 100 次来测试访问 Master 是否工作正常。

```
    # for i in `seq 1 100`;do puppet agent --server puppet.example.com --test;done
```

观察 Apache 的日志，如果 HTTP 状态返回码为 200，表示 Apache 工作正常配置成功。

```
    # tail -f /etc/httpd/logs/access.log
    192.168.110.129 - - [09/May/2014:20:42:14 -0700] "PUT /production/report/bj-web-nginx-1.
example.com HTTP/1.1" 200 14 "-" "-"
    192.168.110.129 - - [09/May/2014:20:42:15 -0700] "POST /production/catalog/
bj-web-nginx-1.example.com HTTP/1.1" 200 921 "-" "-"
```

```
192.168.110.129 - - [09/May/2014:20:42:15 -0700] "PUT
/production/report/bj-web-nginx-1.example.com HTTP/1.1" 200 14 "-" "-"
```

11.3.2 Nginx + Passenger

我们仍然以CentOS 6.5_64位系统为例，介绍如何安装Nginx+Passenger。安装Nginx+Passenger共需要以下5步。

1）软件环境的安装。与Apache+Passenger安装方式比较，除了不需要安装Apache相关软件包外，其他都需要安装。

```
# yum install gcc gcc-c++ curl-devel openssl-devel zlib-devel ruby-devel rubygems
# gem install rack passenger
```

2）运行passenger-install-nginx-module命令，它会帮我们自动完成整个Passenger配置流程。运行过程中它会提示以下两步。

步骤1 自动下载Nginx安装（推荐）。

步骤2 通过已经有的Nginx来安装Passenger。

读者可以根据自己的情况来选择。

```
# passenger-install-nginx-module
# 自动下载 Nginx 安装整合 Passenger（推荐）
1. Yes: download, compile and install Nginx for me. (recommended)
   The easiest way to get started. A stock Nginx 1.6.0 with Passenger
   support, but with no other additional third party modules, will be
   installed for you to a directory of your choice.
# 使用机器自带 Nginx 安装整合 Passenger
 2. No: I want to customize my Nginx installation. (for advanced users)
   Choose this if you want to compile Nginx with more third party modules
   besides Passenger, or if you need to pass additional options to Nginx's
   'configure' script. This installer will  1) ask you for the location of
   the Nginx source code,  2) run the 'configure' script according to your
   instructions, and  3) run 'make install'.
```

3）创建Rack所需要的配置文件与目录。

```
# mkdir -p /etc/puppet/rack{public,tmp}
# cp /usr/share/puppet/ext/rack/files/config.ru /etc/puppet/rack/
# chownn -R puppet:puppet /etc/puppet/rack/
```

4）/etc/nginx.conf配置如下：

```
# 开始
worker_processes  1;
events {
   worker_connections  1024;
}
http {
```

```
    # 设置passenger的环境目录
    passenger_root /usr/lib/ruby/gems/1.8/gems/passenger-4.0.42;
    passenger_ruby /usr/bin/ruby;
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;
# Passenger虚拟主机配置
server {
  listen                          8140 ssl;
  server_name                     puppet puppet.example.com;
  passenger_enabled               on;
   # 设置HTTP头用于验证CA
  passenger_set_cgi_param     HTTP_X_CLIENT_DN $ssl_client_s_dn;
  passenger_set_cgi_param     HTTP_X_CLIENT_VERIFY $ssl_client_verify;
  # 开启Nginx日志，通过此日志监控Nginx服务器的健康状态
  access_log                 /var/log/puppet_access.log;
  error_log                  /var/log/puppet_error.log;
  root                       /etc/puppet/rack/public;
  # 设置Puppet证书文件地址
  ssl_certificate          /var/lib/puppet/ssl/certs/puppet.example.com.pem;
  ssl_certificate_key      /var/lib/puppet/ssl/private_keys/puppet.example.com.pem;
  ssl_crl                  /var/lib/puppet/ssl/ca/ca_crl.pem;
  ssl_client_certificate   /var/lib/puppet/ssl/certs/ca.pem;
  ssl_ciphers                SSLv2:-LOW:-EXPORT:RC4+RSA;
  ssl_prefer_server_ciphers  on;
  ssl_verify_client          optional;
  ssl_verify_depth           1;
  ssl_session_cache          shared:SSL:128m;
  ssl_session_timeout        5m;
}
}
# 结束
```

最后启动Nginx，并通过netstat –tnl命令查看8140端口是否开启。

```
# nginx -t /etc/nginx.conf
# netstat -tnl | grep 8140
```

5）Master守护进程成功启动后，以同样的方式通过for循环测试100次Agent访问Master，以查看是否访问正常。

```
# for i in `seq 1 100`;do puppet agent --server puppet.example.com --test;done
```

观察Nginx的日志，如果HTTP状态返回码为200，则表示Nginx工作正常，配置成功。

```
# tail -f /var/log/puppet_access.log
192.168.110.129 - - [09/May/2014:22:09:22 -0700] "POST /production/catalog/bj-web-nginx-1.example.com HTTP/1.1" 200 921 "-" "-"
192.168.110.129 - - [09/May/2014:22:09:22 -0700] "PUT /production/report/bj-web-nginx-1.example.com HTTP/1.1" 200 14 "-" "-"
```

```
    192.168.110.129 - - [09/May/2014:22:09:24 -0700] "POST /production/catalog/bj-
web-nginx-1.example.com HTTP/1.1" 200 921 "-" "-"
    192.168.110.129 - - [09/May/2014:22:09:24 -0700] "PUT /production/report/bj-web-
nginx-1.example.com HTTP/1.1" 200 14 "-" "-"
```

11.4 Puppet 集群介绍

在前几节中我们已经了解了多种 Master 瓶颈解决方案，这些解决方案目前还是由单机启动多进程，或结合 Web 软件的方式来提升 Master 处理性能。但通过这些方式来管理海量的服务器还是远远不够的，本节将介绍通过之前的技术方案整合，将一台 Master 通过负载均衡技术提供服务变成多台 Master 协作提供服务，也就是 Puppet 集群技术。本节首先从为什么建立集群介绍起，然后介绍建立集群的场景，并结合前几节内容来分析什么样的场景适合建立集群，以及建立集群的形式，最后分析通过负载均衡软件和 DNS 建立集群的利弊。

11.4.1 为什么建立 Puppet 集群

这里以生活中常见的火车票购票为例，来解释为什么要建立 Puppet 集群。在生活中我们都买过火车票，特别是在早些年互联网还不是很发达的时候，通常是到火车站的售票大厅买票。进入火车站售票大厅后我们会看到很多的售票窗口（可以将窗口看为 Master 端，主要任务是提供服务），售票窗口的用途是为旅客提供售票的服务（可以将旅客看为 Agent 端，主要任务是从 Master 获取信息）。我们买火车票时也经常会遇见这样的问题，当旅客比较多的时候，一个窗口无法满足旅客的购票需求，导致队尾旅客也越来越多，购票的队伍越来越长，等待的时间也会越来越长，同时也对售票窗口造成很大的压力。这时售票大厅就要合理地通过增加售票窗口的方式，缩短购票时间，让等待买票的旅客尽快买到车票，让排队的旅客等待的时间变短，这样不但缓解了旅客因等待而产生的烦躁情绪，提高旅客的满意度，同时也解决了售票窗口的压力。建立 Puppet 集群与售票大厅的原理一样，当访问 Master 的 Agent 比较多时，就会导致 Master 很忙，处理请求会变得越来越慢，甚至会导致 Master 假死，最终使得整个服务不可用。这时就需要通过追加更多 Master 的方式来均衡负载 Agent 的访问请求，保证整个服务的可用性，这也是我们通过 Puppet 建立集群的真正目的。

11.4.2 建立 Puppet 集群的场景

了解了通过 Puppet 建立集群的目的后，再来看一下什么情况下适合建立 Puppet 集群。如果工作场景中只有 10 台以内的 Agent 是否要建立 Puppet 集群呢？答案是否定的。在介绍 Puppet 使用场景之前，先介绍一下使用 Puppet 的机器配置要求、工作原理和瓶颈问题。只

有了解了这些我们才能知道在什么场景下适合建立 Puppet 集群，来解决我们运维工作中遇到的问题。

笔者在第 2 章中曾介绍过 Puppet 的最低配置，这里再来回顾一下。Puppet 官方网站建议使用 Master 的最小机器配置是双核 CPU，1GB 内存；推荐配置 2~4 核 CPU，4GB 以上内存配置，这样的配置大约可以满足管理 1000 个 Agent。Master 的工作原理是通过 WEBRick HTTP 服务（一款 Ruby 内置的 Web 服务器）接收 Agent 的请求。WEBRick HTTP 服务本身的处理能力有限，并且它是一款轻量级单进程处理请求的 Web 服务器，并不适合大量的并发访问，这也是通过 Puppet 管理配置海量服务器的短板之一。

再来看一下 Puppet 在实际使用中遇到的瓶颈。其实刚刚在介绍 Master 的访问原理时就介绍到 Puppet 通过 WEBRick HTTP 服务来处理 Agent 请求时，Master 本身的 WEBRick HTTP 服务的处理能力就是一种瓶颈。另外 Puppet 是配置管理工具，其主要目的是用来配置管理 Agent 的系统文件，在同步配置文件时，并不适合同步比较大的数据文件，因为 Puppet 并非专业的文件同步工具，当从 Master 同步较大文件时会导致 Agent 获取配置信息超时，最终导致整个访问获取配置信息的失败，这也是它的瓶颈之一。最后就是网络瓶颈，如在笔者的工作环境中，根据业务的重要程度会将它分别部署在多地的多个 IDC 上，目的是保证业务容灾与高可用性。同时多 IDC 部署还可以让用户就近接入，提高用户体验。通过 Master 管理 Agent，当这些 Agent 与 Master 不在同一个 IDC 时，部分 Agent 会由于跨网的原因导致超时，最终获取不到 Master 的配置信息，这也是网络架构的瓶颈导致的问题。这些都是使用 Puppet 管理配置经常遇见的瓶颈，需要具体问题具体分析，来看如何解决这些问题。

最后再来看一下适合建立 Puppet 集群的场景：

- 机器大于 1000 个 Agent 的场景。
- 通过 Puppet 同步大量的配置文件分发的场景。
- 跨 IDC 访问 Master 的场景。
- Puppet SSL 认证频繁超时的场景。
- 通过 Puppet 来管理云端服务器，如 Openstack、Eucalyptus Cloud、Amazon EC2 和 Google Compute Engine 等场景。

11.4.3 集群负载均衡解决方案

集群的负载均衡解决方案包括常见的两种，一种为通过代理机接收访问并按策略分发这些访问到后端机器，另一种为通过 DNS 轮训技术来分发机器的访问。两种方式各有利弊，本节将来介绍这两种常见的技术解决方案。

1. 负载均衡技术

通过负载均衡技术来建设 Puppet 集群。负载均衡技术的优势是可以检测后端机器状态，

当机器状态不可用时（不可用包括：机器死机、硬件故障、网络故障和进程异常等），负载均衡技术可以不发送请求给后端不可用的服务器，保障服务的可用性。负载均衡技术中包含常见的 HAProxy、LVS、BigIP、Nginx 和 Apache 等。以 LVS 为例，如图 11-1 所示，Puppet 可以通过 LVS 中的 VIP（virtual IP，中文译为虚拟 IP）技术来做负载均衡访问，后端的 Master 可以根据 Agent 访问量平行的扩展 Master。如果 LVS 本身出现故障导致服务不可用，我们还可以与 HAProxy 结合实现 Puppet 服务的高可用。此技术适合节点大于 1000 台服务器的使用场景。

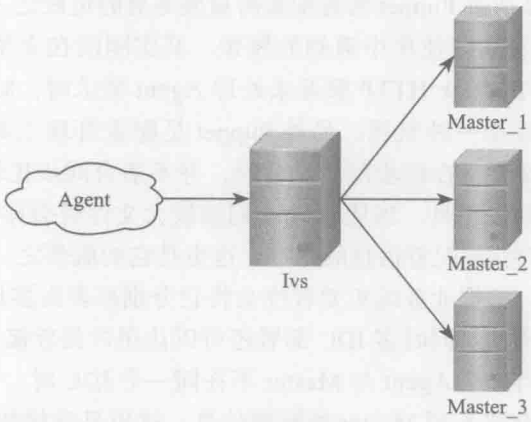


图 11-1 LVS 架构图

2. DNS 技术

除了使用负载均衡技术外，还可以通过 DNS 轮询与 DNS VIEW（DNS 的“就近解析”）技术来解决 Master 的处理瓶颈与网络瓶颈。首先来了解一下 DNS 轮询技术，它主要用来解决 Master 的处理瓶颈。DNS 轮询技术可以将 Agent 访问随机地发送到后端的 Master 上处理，从而解决 Master 处理瓶颈。但 DNS 轮询技术也有它的弊端，DNS 轮询不会检测后端 Master 的状态，当某台 Master 状态发生故障或者下线时，DNS 轮询依然会将 Agent 请求转发给发生故障或下线的 Master，最终导致 Agent 获取配置信息失败。在使用 DNS 轮询时，也需要注意避免由机器下线与故障的情况带来的服务不可用的情况。


注意 使用 DNS 轮询技术时，建议后端采用单独的 Puppet CA 认证服务器或采用稍后介绍的单证书扩展解决方案。在 DNS 轮询技术方案中使用单独的 CA 认证服务器，Agent 端的配置需要设 --ca_server 参数。

最后来了解一下 DNS VIEW 技术，它主要用来解决网络瓶颈（由于不同的机房可能不在同一个网络中，通过公网传输数据就会出现延迟或丢包等常见网络现象，从而影响 Agent 与 Master 的正常同步数据，这种情况统称为网络瓶颈）。当 Master 分布在不同的 IDC 时，

就可以通过DNS VIEW技术根据Agent源或IP网段来实现就近解析功能，如图11-2所示。在每个IDC部署相同逻辑的Puppet Master，让不同IDC之前间的Agent通过DNS VIEW技术解析到本IDC的Master，通过此方式可绕开网络瓶颈对Agent与Master同步数据的影响。

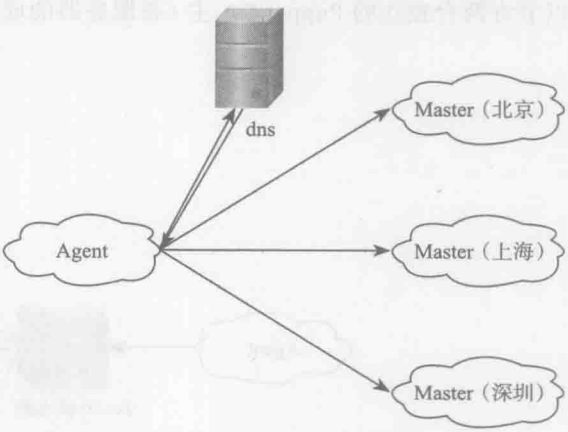


图11-2 DNS VIEW架构图

11.5 Puppet CA均衡负载

Puppet CA均衡负载解决方案用于Puppet集群中SSL集中认证场景，我们可以通过CA认证服务器与Puppet配置服务器进行分离来提高CA的处理性能，也可以在所有的Master使用单独的证书来进行认证。两者对比笔者更推荐使用单独证书认证的方式，因为它更加节约成本。

1. 认证解决方案

在建立Puppet集群后，由于证书之间存在序号与吊销列表等问题，通常将Agent关于认证部分的请求发送到单独的Puppet CA服务器来集中处理。更改Agent认证请求可以使用以下两种方式。

方式1：Agent端通过puppet agent --server puppet.example.com --test --ca_server puppetca.example.com方式来指定统一的认证服务器。

方式2：Agent端修改puppet.conf配置文件，增加以下内容。

```
[agent]
masterport = 8140
environment = production
server = puppet.example.com
ca_server = puppetca.example.com
```

但需要注意的是，Puppet CA服务器在整个流程中处于很重要的一个环节，如果Puppet CA服务器出现故障则会导致整个Puppet的服务不可用，所以要做好Puppet CA服务器的灾备，通过双机灾备来提升服务的可用率，如图11-3所示。

2. 单证书认证扩展解决方案

在建立Puppet集群之初，由于证书之间存在序号与吊销列表等问题，我们不得不由一台独立的Puppet CA来提供Agent认证等服务。下面再来介绍一种解决方案，将单独的证

书平行分发到其他的 Master 上共同使用。笔者推荐使用单证书平行扩展的方式，因为它可以节省两台独立的 Puppet CA 主 / 备服务器的成本。

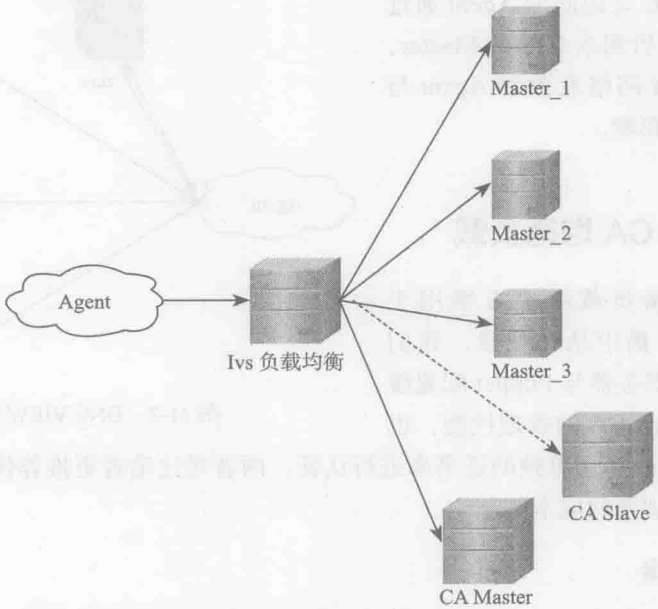


图 11-3 Puppet CA 容灾架构图

首先在已经配置好并生成了证书文件的 Master 服务器上将证书目录打包。以笔者的机器运行环境为例，具体步骤如下：

1）将 puppet.conf 配置文件中 ssldir 参数的目录路径进行打包。

```
# tar -cvzf  server_ssl.tar.gz  /etc/puppet/server_ssl
```

2）将 server_ssl.tar.gz 文件同步到其他 Master 服务器上并进行解压。

```
# Master1(IP:192.168.1.1): Scp server_ssl.tar.gz  root@192.168.1.2:/etc/puppet/
# Master2(IP:192.168.1.2): tar -xvzf /etc/puppet/server_ssl.tar.gz
```

3）在 Master2 上重新启动 Pupppetmasterd，并测试 Master 是否正常工作。

```
# service puppetmasterd restart
```

第 12 章 Chapter 12

报告系统

在我们通过 Puppet 管理海量服务器的时候，一个配置变更发布到服务器上最终执行是否能成功？整个配置变更耗时是多少？最终的变更状态如何？这些都是我们通过 Puppet 运维海量服务器需要了解的核心指标。而 Puppet 为我们提供了这样一款的工具，它可以直观地告诉我们目前线上服务运行的状态，可以与报告处理器结合提供图形化展示、报表展示，并且在出现问题的情况下以邮件形式进行通知和系统告警等，让我们对线上服务的运营状态一目了然。如果这些还不能满足我们的需求，它还可以通过 Ruby 语言定制上报数据格式，让数据展示更加个性化与实用。另外 Puppet 还可以对运营数据（如变更、重启、失败和跳过等数据）进行沉淀，通过对沉淀后的数据进行分析，可对不合理之处进行优化与改进。这样一款方便实用的工具便是 Puppet 的报告系统。

12.1 报告系统入门

Agent 运行时会生成一份运行状态的报告，并通过 Puppet::Transaction::Report 类生成 YAML 格式从 Agent 推送到 Master 的指定目录下，并由 Master 进行汇总展示，从而使得运维工程师可以通过 Master 的汇总展示了解系统的运营状况。以 Agent（Hostname 为 agent.web.puppet.example.com）为例，它访问 Master 后就会在 /var/lib/puppet/reports/agent.web.puppet.example.com/ 目录下找到这份报告，这份报告以时间戳为文件名，以 .yaml 为扩展名。下面我们实际操作一次从 Agent 上报一份报告到 Master 上。需要读者注意，如果是 Puppet 2.7 以上的版本，Agent 默认会推送报告到 Master 的 /var/lib/puppet/reports/ 目录中，如果是 Puppet 2.7 以下的版本，需要通过增加执行参数或修改 puppet.conf 配置文件的方式

打开报告上报功能。

注意 我们可以通过 Master 的 puppet.conf 中的 reportdir 参数来修改报告存放的默认位置。

Puppet 2.6 版本通过 --report 参数，开启 Agent 上报功能的方式。

```
# puppet agent --server example.com.com --test --report
```

Agent 通过 --report 参数可以打开报告上报的功能。另外还可以通过修改 Agent 的 puppet.conf 配置文件，增加 report=true 方式来打开报告上报功能。具体的打开方法如下：

```
[agent]
report = true
```

我们以命令方式为例，在 Agent 通过执行 puppet agent --server example.com.com --test --report，就会在 Master 目录中找到 Agent（agent.web.example.com）上报的报告。报告内容如下：

```
# cat /var/lib/puppet/reports/agent.web.example.com/201404021107.yaml
--- !ruby/object:Puppet::Transaction::Report
  report_format: 3
  resource_statuses:
    Schedule[monthly]: !ruby/object:Puppet::Resource::Status
      resource: Schedule[monthly]
```

由于报告的内容比较多，所以对以上的内容作了截取。收到的 Agent 报告主要包含以下内容：

- 清单中的资源和 tag 列表。
- 清单中的资源列表、成功状态和变更列表。
- Puppet 整体运行耗时、Puppet 版本号、报告上报时间、上报格式等。

通过以上报告截取的输出可以了解到报告内容还是比较乱的，在不借助工具的情况下比较难看懂。其实可以通过 Agent 的 summarize 参数查看到主要的汇总信息，这样看起来就比较清晰了。

```
# puppet agent --server puppet.example.com --test --summarize
Events:
Resources:
          Skipped: 6          # 跳过资源
            Total: 8          # 总共资源
Time:
       Filebucket: 0.00       # 备份耗时
             File: 0.00       # 文件耗时
  Config retrieval: 0.41      # 配置耗时
            Total: 0.41       # 耗时
```

```
    Last run: 1396529148        # 最后运行时间
Version:
        Config: 1396529148      # 上报时间
        Puppet: 2.7.21          # 上报版本号
```

注意 如果Master的后端中采用了均衡负载技术，我们可以修改Agent的puppet.conf中的reportserver，将上报信息汇总到一台集中的服务器进行存储，这也便于后续的数据分析。

12.2 报告处理器

报告处理器的作用是将Agent上报的日志根据用户的需求进行展示或告警。

通过命令方式可以开启报告器。具体的开启命令如下：

```
# puppet master --reports log,tagmail
```

通过修改puppet.conf方式也可以开启报告器。具体的开启方法如下：

```
[master]
reports = store,log,tagmail,rrdgraph,http
```

下面简单分析一下上面常见报告器的作用。

- store：默认参数，将报告存到磁盘上。
- log：将报告发送到系统的syslog。
- tagmail：通过邮件形式发送报告给接收者。
- rrdgraph：它利用Tobias Oetiker的RRD图形库生成图形来展示报告状态。
- http：将报告POST到指定的URL，通常与Dashboard一起使用。

这里store默认参数就不再介绍了，如果开启多个报告器可以“,”作为分隔。下面分别来看log、tagmail、rrdgraph和http方式的报告处理器的使用方式。

（1）log报告处理器

log报告处理器会将Agent的每一条报告发送到本机的syslog。并可以通过syslogfacility参数来控制发送的设备用户。形式如下：

```
[master]
reports = store,log
syslogfacility = user
```

log报告器将Agent上报的报告写入系统的syslog，其优势是系统的syslog有很多统计与分析工具，借助这些工具可以方便地分析出日志中存在的问题，降低日志分析与统计的成本。

（2）tagmail 报告处理器

tagmail 处理器可以将报告通过邮件的形式发送给管理员，每一条邮件中包含了原始报告的标签，标签可以帮助我们联系上下文。已经配置好的 tagmail 报告处理器的工作方式如下：

```
# puppet agent --test
info: Caching catalog for master.hightower.org
info: Applying configuration version '1314140379'
notice: Finished catalog run in 0.02 seconds
```

执行后我们会收到以下的邮件：

```
From: report@master.hightower.org    # 通过 tagmail.conf 配置发件邮箱
Subject: Puppet Report for master.hightower.org     # 邮件标题
To: kelsey.hightower@gmail.com     # 通过 tagmail.conf 配置发送给谁
Message-Id: 20110823225940.363555E87B@master.hightower.org     # 邮件标签
Date: Tue, 23 Aug 2011 18:59:40 -0400 (EDT)
Tue Aug 23 18:59:39 -0400 2011 Puppet (info): Caching catalog for master.
hightower.org
Tue Aug 23 18:59:39 -0400 2011 Puppet (info): Applying configuration version
'1314140379'
Tue Aug 23 18:59:39 -0400 2011 Puppet (notice): Finished catalog run in 0.02
seconds
```

通过 tagmail 处理器实现以上的邮件发送形式，需要打开 puppet.conf 的配置。开启方式如下：

```
[master]
reports = store,tagmail
reportfrom = webmaster@example.com
tagmap = $confdir/tagmail.conf
```

rrepots 参数用于设置开启报告处理器，tagmail 需要通过 sendmail 将报告以邮件形式发送，这里需要配置 tagmail.conf 的配置文件。tagmail 配置文件内容如下：

```
all: puppeter@example.com
db,!mail:app@example.com
err: puppeter_err@example.com,puppet_err_user@example.com
```

如以上配置所示，all、db、err 为特殊的标签，用于通知 Puppet 将哪些邮件报告发送给谁，并通过“：”指定发送的邮箱地址。all 标签用于通知 Puppet 将所有的报告发送给指定的 puppeter@example.com 邮箱。db 标签中包含了“!”，用于禁止将邮件报告发送到指定的邮箱，db 标签会将与 db 有关的邮件禁止发送到 app@example.com 邮箱。最后的 err 标签表示日志的级别，它将配置运行过程中的错误信息发送到 puppeter_err@example.com 和 puppet_err_user@example.com 邮箱。除此之外，日志级别还包含了 debug、info、notice、warning、alert、emerg、crit 和 verbose 等。建议开启 err 日志，其他日志可以选择性开启，因为当我

们管理的服务器比较多的时候，所有日志都开启就会掩盖重要的错误日志信息，使发现问题的及时性大打折扣。

（3）rrdgraph 报告处理器

rrdgraph 是一个非常实用的报告处理器，它可以生成 RRD 图形文件，并以这种更直观的方式生成图形化的报告，报告的信息主要包含事件（events）、资源（resources）和获取事件（retrieval time）等，这种简单快速的方式可以让我们了解 Agent 的运行状况。

要想通过 rrdgraph 报告器生成漂亮的图形报告，还需要安装 RRDTools 和 RRD 的 Ruby 绑定环境，rrdgraph 报告器通过它们来生成 RRD 图形文件。但不幸的是 RRD 的 Ruby 绑定对许多平台的支持状况并不好，除了一些基于 RPM 平台的 RedHat 和 CentOS 发行版本外，其他的平台需要通过源码编译的方式来安装。以下是以 CentOS 6.5_x86_64 位系统为例的安装方式。

```
# yum install rrdtool rrdtool-ruby
```

在 Debian/Ubuntu、Fedora 和 RedHat 系统安装时需要注意安装软件包名，如表 12-1 所示。

表 12-1 rrdtools 软件包名

操作系统	软 件 包
Debian/Ubuntu	rrdtool、librrd2、librrd2-dev
Fedora	rrdtool、rrdtool-ruby
RedHat	rrdtool、rrdtool-ruby

其他发行版本可以通过源码编译的方式来安装 RRD 的 Ruby 绑定环境。源码编译方式如下：

```
# wget http://rubyforge.org/frs/download.php/13992/RubyRRDtool-0.6.0.tgz
# tar -xvzf RubyRRDtool-0.6.0.tgz
# cd RubyRRDtool-0.6.0
# ruby extconf.rb
# make
# makein stall
```

RRD 环境安装好后，接着修改 Master 的 puppet.conf 配置文件，添加以下的配置：

```
[master]
reports = store,rrdgraph
```

加入配置后重新启动 Master 的守护进程。Agent（agent.web.example.com）访问 Master 后，就会看到以下的内容：

```
# ls /var/lib/puppet/rrd/agent.web.example.com/
changes-daily.png    events.rrd           resources-yearly.png
changes.html         events-weekly.png    time-daily.png
```

由于目录的内容比较多，所以笔者对以上内容作了截取。Puppet 开启 rrdgraph 报告器后，Agent 访问 Master 就会在 /var/lib/puppet/rrd/ 目录下生成以 Agent 的 Hostname 为名的目录，目录中包含了很多的 HTML 文件、RRD 文件和 PNG 图像文件。其中 HTML 文件主要与 Web 服务器结合浏览生成的报告所使用，RRD 为图形文件的源数据文件，PNG 为生成后的图像文件，rrdgraph 会定期通过 RRD 图形源更新 PNG 图像。我们可以通过 Web

服务器的虚拟目录的方式来进行浏览。以 Apache 为例，增加虚拟目录的配置到 httpd.conf 尾部。

```
Alias /rrd/ "/var/lib/puppet/rrd/"     # 指定 Puppet 的 rrdgraph 报告器的根目录
<Directory "/var/lib/puppet/rrd">      # 指定 Puppet 的 rrdgraph 报告器的目录，设置目录的权限
Options Indexes MultiViews
AllowOverride None
Order allow,deny
Allow from all
</Directory>
```

修改后重启 Apache（service httpd restart），最后可通过浏览器来进行查看，如图 12-1 所示。

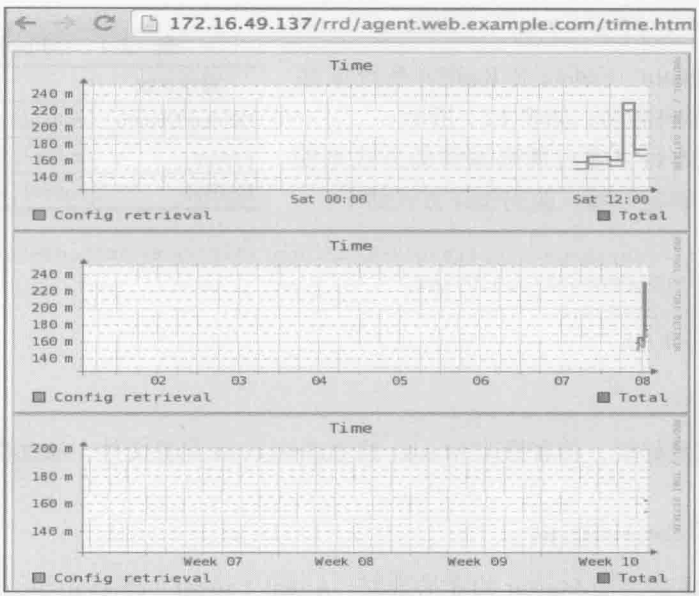


图 12-1 rrdgraph 报告器生成的图形

（4）http 报告处理器

http 报告处理器通常与 Dashboard 结合使用，我们会在下一章中介绍 Puppet 的 Dashboard。http 报告器通过 HTTP 协议将 YAML 格式数据，以 POST 的形式转发到一个 http 或 https 的 URL 进行存储。修改 Master 的 puppet.conf 中的 reporturl 参数来配置报告的目标发送地址。具体如下：

```
[master]
reports = store,http
reporturl = http://localhost:3000/reports     # 发送到本机的 Dashboard
```

12.3 自定义报告处理器

如果系统自带的报告处理器不能满足我们的需求，还可以通过 Puppet 创建自定义报告处理器。目前 Puppet 创建自定义处理器的常见方式有以下两种。

方式 1 使用目前已经存储在服务器上的 YAML 格式源文件，并根据源文件进行分析处理绘制图形或导入数据库进行下一步的分析，这也是 Dashboard 中报告输入进程的工作原理。

方式 2 与第 10 章中介绍的扩展自定义函数、类型和提供者一样，通过 Ruby 语言来创建自定义处理器，并使用 puppet.ocnf 中的 pluginsysnc 参数同步自定义处理器。

下面以方式 2 为例来介绍如何创建自定义处理器。在第 10 章笔者曾强调过在通过 Ruby 语言扩展新的功能前，最好先阅读一下目前已有的处理器源码，对现有的处理器原理有了基本的了解后再来创建自定义处理器。下面以 log 处理器为例来对其代码与目录结构进行分析，然后再来介绍如何创建自定义报告处理器。

12.3.1 log 处理器源码分析

首先我们需要找到 Puppet 自带的处理器的位置，根据 Puppet 安装方式不同，源码目录的存放位置也不一样。可以通过 Facter 工具来查找 rubysitedir 环境变量的位置，最终找到处理器的源码目录。具体的实现命令如下：

```
# facter | grep rubysitedir
rubysitedir => /usr/lib/ruby/site_ruby/1.8/
```

通过以上 facter 命令输出可以查到 Puppet 的安装目录，通过输出了解到它是 yum 默认的安装方式，Puppet 的安装目录在 /usr/lib/ruby/site_ruby/1.8/puppet 下。Puppet 安装位置找到了，那么处理器的源码位置就在 $rubysitedir/puppet/reports 目录下。以下为 reports 目录中的报告处理器源码文件。

```
# ls /usr/lib/ruby/site_ruby/1.8/puppet/reports
http.rb  log.rb  puppetdb.rb  rrdgraph.rb store.rb tagmail.rb
```

以下为 log.rb 的源码程序内容：

```
require 'puppet/reports'     # 加载 reports 的库文件
Puppet::Reports.register_report(:log) do     # 声明 log 处理器（log 要与文件名一致）
  desc "Send all received logs to the local log destinations. Usually  # 处理器的描述文档
    the log destination is syslog."
  def process      # 定义处理逻辑
    self.logs.each do |log|     # 将运行日志遍历到 log 中
      log.source = "//#{self.host}/#{log.source}"     # 追加客户端主机名信息
      Puppet::Util::Log.newmessage(log)     # 通过 Puppet::Util::Log.newmessage 方法发送到 syslog
```

```
  end
 end
end
```

如以上代码所示，Puppet 通过 Puppet::Reports.register_report 来声明处理器，在代码中我们会看到很多的 self.*，它们主要用来提取信息，可以在表 12-2 中找到它们的含义。

表 12-2 self 方法定义

方 法 名	含 义	方 法 名	含 义
self.logs	日志	self.status	运行状态
self.host	节点主机名	self.everonment	客户端环境
self.configuration_version	配置的版本信息	self.to_yaml	转为 YAML 格式输出
self.metrics	度量值		

12.3.2 自定义报告处理器

了解了系统自带的 log 报告处理器目录与逻辑后，我们来创建自己的自定义报告处理器，其功能是将一些摘要信息写入 summary.txt 文件中。首先创建自定义报告处理器的目录结构，并介绍自定义报告器开发过程中的注意事项。

```
# mkdir -p /etc/puppet/modules/custom/{manifests lib}
# mkdir -p /etc/puppet/modules/custom/lib/puppet/reports/
# 创建后的目录结构
/etc/puppet/modules
└── custom
    └── manifests
    |       └── init.pp     # 自定义模块加载文件
    └── lib
        └── puppet
            ├── reports
            |   └── summary.rb     # 自定义处理器源码文件
```

如以上目录结构所示，创建的 manifests 目录主要用于存放 init.pp 文件，此文件为用户模块的初始化加载，如果无初始化内容，可以声明空类，否则会报错。编辑 init.pp 文件内容如下：

```
# /etc/puppet/modules/custom/lib/puppet/manifests/init.pp
class custom{
}
```

接着创建 reports 目录，用于存放自定义处理器的源码文件。在创建自定义报告处理器前，需要注意以下的事项：

- 处理器的名字可以包含字母和数字，并且建议名字以字母作为开头。
- 处理器源码需要用 Ruby 语言编写，并且声明的处理器名需要与源文件名一致。

```
# /etc/puppet/modules/custom/lib/puppet/lib/summary.rb    # 文件名 summary.rb
Puppet::Reports.register_report(:summary) do       # 声明的处理器名 summary
    # 内容省略
end
```

- 处理器源码的头部需要加载 require 'puppet' Puppet 库文件。
- Puppet 通过 Puppet::Reports.register_report() 方法来声明处理器，声明过程中不能包含任何的参数。处理器内部还要包含 desc 方法和 process 方法，desc 方法用来描述处理器的作用与使用方法，process 方法承载了处理器的整个逻辑处理部分。
- Master 开启报告器处理功能后，Agent 访问 Master 就会根据报告器设置生成一份新的报告，并将该报告存放在 /var/lib/puppet/reports/Hostnmae/ 目录下。

接着编辑 summary.rb 自定义报告器，并追加以下内容到源文件中。

```
# /etc/puppet/modules/custom/lib/puppet/lib/suumary.rb
require 'puppet'     # 必须加载的 Puppet 头文件
Puppet::Reports.register_report(:suumary) do   # 创建自定义处理器，声明的处理器名需要与文件名一致
  desc <<-DESC     # desc 方法
   send summary report
  DESC
  def process     # process 方法
    client = self.host
    summary = self.summary
    dir = File.join(Puppet[:reportdir],client)
    client = self.host
    file = "summary.txt"
    destination = File.join(dir,file)
    File.open(destination,"w") do |f|
      f.write(summary)
    end
  end
end
```

通过 ruby -c summary.rb（-c 是 Ruby 的语法检查参数）检查 Ruby 语法是否正确。确认语法没有问题后编辑 Master 的 puppet.conf 文件，追加以下内容到文件中。

```
[main]
pluginsync = true          # 开启扩展同步功能
[master]
reports = store,summary    # 打开 store 默认处理器与 summary 自定义处理器
```

编辑 puppet.conf 配置文件后，需要重启 Master 的守护进程。最后通过 puppet agent --server puppet.example.com --test 来验证结果。如果测试成功 Master 会在报告目录中生成一份 summary.txt 文本文件汇总信息。

12.3.3 个性化处理器

除了 Puppet 自带的处理器和自定义处理器外，Puppet 官方网站上还有很多热心网友提供的个性化处理器，这些处理器可以将报告推送到手机或 IPAD 等一些移动智能设备上。以笔者的工作环境为例，可以将处理器的数据格式化后上报到手机客户端，并通过手机客户端做图形展示，让我们充分感受移动互联网带来的新体验，如图 12-2 所示。

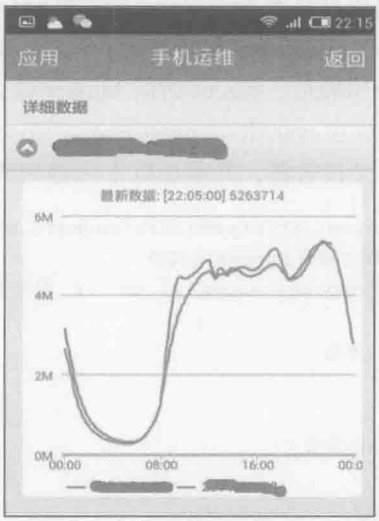
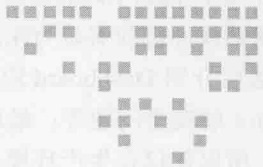


图 12-2 手机报告展示处理器

读者可能会有疑问，Puppet 的处理器将数据上报到手机客户端，那我们是否也需要了解手机客户端的开发呢？答案是否定的，因为现在国内很多互联网公司，如腾讯、新浪，都提供了各种手机型号的客户端，我们可以借助这些客户端来推送信息。以下是腾讯与新浪微博的接口开发文档。

- 腾讯：http://wiki.open.t.qq.com/index.php/%E9%A6%96%E9%A1%B5。
- 新浪：http://open.weibo.com/wiki/%E9%A6%96%E9%A1%B5。

我们可以借助开发文档，将 Agent 相关的错误报告（如果推送所有的报告，信息量会很大，所以建议大家只推送错误报告）推送到手机上。除此之外还可以使用 Puppet 官网已经开发好的处理器，如推送 IRC 处理器、推送 Twitte 处理器、推送 Hip-chat 处理器和推送 Growl 处理器等。这些处理器可以在 http://docs.puppetlabs.com/guides/reporting.html 找到，就不再详细介绍。



第 13 章 Chapter 13

Puppet Web GUI

在之前我们掌握的知识体系中，还是使用清单的方式来管理配置 Agent，这一方式需要我们登录到服务器调整清单规则来配置 Agent。但随着 Puppet 的日渐成熟与完善，一些围绕 Puppet 的 Web GUI 开始出现，它们形成了一套小的生态系统，使我们无须再登录服务器而直接在 Web 界面上管理配置 Agent。比较常见的 Web GUI 有 Enterprise Management System（Puppet 企业版 Web 管理系统）、Foreman 和 Puppet Dashboard（Puppet 控制台）系统等。Enterprise Management System 的优势是企业可以不设置专业的运维工程师维护系统，通过购买 Enterprise Management System 官方服务对企业提供专业的技术支持，劣势自然是它是收费的系统。Foreman（http://theforeman.org）是 Ruby on Rails 程序，它是一个集成的数据中心生命周期管理工具，提供了服务开通、配置管理以及报告等功能。Foreman 的优势是功能比较丰富而且免费，但劣势是一些功能的配置比较复杂，不如 Puppet Dashboard 简单明了。Puppet Dashboard[⊖]由 Puppet 官方提供维护，与 Foreman 一样也是 Ruby on Rails 程序，它的优势是安装与操作简单方便，适用于报告系统的数据展示与部分的 ENC 管理，劣势是功能并不强，常常需要借助清单功能。这 3 款 Web GUI 各有利弊，对于大部分企业来说没有一款 Web GUI 完全适用于企业内部网需求，只能是取长补短。所以本章只介绍 Puppet Dashboard，让读者了解它的安装、配置与使用流程，建议有开发能力的读者可以结合本章对 Puppet Dashboard 的介绍，并借助一些开源的框架（前台框架推荐 Bootstrap 与 Easyui，后台框架推荐 Doitphp 与 PHP CI 等）独立开发更适合自己的 Web GUI，这也不是什么难事。关于 Web GUI 的独立开发，不同的人有不同的需求，鉴于篇幅的原因不在这里详细介绍。

⊖ 参见 http://docs.puppetlabs.com/dashboard/manual/1.2/。

本章首先介绍 Puppet Dashboard 的安装与升级，读者需要关注 Puppet Dashboard 安装过程中的一些辅助软件包的安装与注意事项；接着介绍配置 Puppet Dashboard，通过 4 步让它启动起来；然后介绍 Dashboard 应用场景，同时也包含了人们经常关注的问题的解决方法，如 Dashboard 的安全问题等；最后介绍 Dashboard 与 Nginx 提升性能，由于 Webrick 的性能并不强劲，所以建议在生产环境中通过 Nginx 将 Webrick 替换为 Web 服务器。

13.1 Puppet Dashboard 安装与升级

Agent 会定期上报日志到 Master，由于我们管理的 Agent 比较多，少量的 Agent 出现故障时，快速定位出现故障的 Agent 是摆在运维工程师面前的一道难题。而 Puppet 正巧提供了一款工具来解决这样的问题，它将数据可视化地建立了数值与人的连接，借助图形的力量，清晰有效地展示了问题的所在，这款工具就是 Puppet Dashboard。Puppet Dashboard（中文译为“Puppet 控制台”，下称 Dashboard）由官方网站提供开发与维护。Dashboard 是基于 Ruby on Rails 的 Web 程序，可以运行在大多数的 UNIX/Linux 系统平台（包含 Mac OS 系统）。Dashboard 适合海量服务器的管理场景，它的优势是可以通过 Web 方式查看现有 Master/Agent 的工作状态，通过 ENC 方式从 Web 端来管理 Puppet 节点和统计分析 Agent 上报的日志等。

笔者以安装 Dashboard-1.2.23 版本为例来介绍整个 Dashboard 的安装过程（此版本为 2013 年 5 月 14 日更新的版本，截止到本书发行前此版本没有再更新，可见此版本已经基本稳定，没有严重的漏洞。另外 Puppetlabs 也有意向让用户向 Puppet 企业收费版本转移）。在安装 Dashboard 前需要预先安装好相关依赖环境，包括 Ruby1.8.7 版本（Dashboard 对 Ruby1.9.2 的支持并不完全）和用于存储数据的开源关系型数据库 MySQL[⊖]，目前 Dashboard 只支持 MySQL 作为后端存储，相信不久的将来 Puppet 会支持更多的数据库系统。除了 Ruby 和 MySQL 软件包之外 Dashboard 还需要安装以下软件包支持。

- RubyGems：它是一个库和程序的标准化打包以及安装框架，使得定位、安装、升级和卸载 Ruby 包变得更加容易。
- Rake 0.8.3 或更高版本：Rake 是一门构建语言，与 Make 很相像。它是用 Ruby 编写的，支持它自己的 DSL 用来处理和维护 Ruby 应用程序。
- Ruby-MySQL 2.7.* 或 2.8.* 版本：MySQL 数据库连接驱动，用于建立 Puppet 与数据库的连接。

1. 在 RedHat 上安装 Dashboard 需要的辅助软件包

如果在我们之前安装过 Master 的服务器上安装 Dashboard，可以通过以下的 yum 直接

⊖ 原开发者为瑞典的 MySQL AB 公司，该公司于 2008 年被 SUN 公司收购。2009 年，甲骨文公司又收购了 SUN 公司，所以目前 MySQL 成为 Oracle 旗下产品。

安装这些辅助的软件包。

```
# yum install  rubygems  rubygem-rake  mysql-server mysql-devel ruby-mysql
```

如果是单独搭建新的 Dashboard 服务器，除以上安装包外，还需要追加以下的安装包：

```
# yum install  ruby-1.8.7 ruby-devel -ruby-irb ruby-rdoc ruby-ri
```

另外如果我们使用比较老的操作系统发行版本，如 RedHat 以及 CentOS 的 5.x 版本，yum 安装的 RubyGems 包管理系统还不太合适，需要通过源码的方式进行下载安装。以下为官方网站提供的安装脚本，编辑 install_rubygems.sh，将以下内容追加到脚本中。

```
#!/bin/bash
URL="http://production.cf.rubygems.org/rubygems/rubygems-1.3.7.tgz"    # rubygems
1.3.7 下载地址
PACKAGE=$(echo $URL | sed "s/\.[^\.]*$//; s/^.*\///") # 匹配到 rubygems-1.3.7.tgz 包名
cd $(mktemp -d /tmp/install_rubygems.XXXXXXXXXX) && \    # 创建临时目录
wget -c -t10 -T20 -q $URL && \    # 下载 rubygems1.3.7 软件包
tar xfz $PACKAGE.tgz && \    # 解压安装包
cd $PACKAGE && \
sudo ruby setup.rb    # 安装 rubygems 软件包
```

执行命令 sh install_rubygems.sh 安装脚本，即可安装 RubyGems 软件包环境。

2. 在 Ubuntu 上安装 Dashboard 需要的辅助软件包

在 Ubuntu 10.0.4 以上版本，可以通过 apt-get 安装以下的软件包：

```
# apt-get install -y build-essential irb libmysql-ruby libmysqlclient-dev \
libopenssl-ruby libreadline-ruby mysql-server rake rdoc ri ruby ruby-dev
```

需要注意的是，它与 RedHat 系统存在同样的问题，RubyGems 并不太适合 Ubuntu 10.0.4 之前的版本，所以这里需要单独安装 RubyGems。在 Ubuntu 上安装 RubyGems 的方式与 RedHat 系统一样，需要通过源码方式安装，读者可以参考 RedHat 中的 RubyGems 安装脚本，这里不再单独介绍。在通过源码安装 RubyGems 后，需要通过 update-alternatives 命令添加最新的 RubyGems 版本作为默认的 Gem 管理器。添加方式如下：

```
# update-alternatives --install /usr/bin/gem gem /usr/bin/gem1.8 1
```

3. 安装 Dashboard

Dashboard 基础环境安装后，我们来安装 Dashboard。安装 Dashboard 的方式比较灵活，可以通过包仓库来安装，也可以通过源码方式来安装，还可以通过 GIT 来安装。安装后 Dashboard 的程序默认会存放在 /usr/share/puppet-dashboard 目录中。以下为 Dashboard 的 3 种安装方式。

1）包仓库安装方式（推荐）：通过 yum 方式来安装 Dashboard，适用于 RedHat 和

CentOS 发行版本。

首先需要安装 Puppet 提供的 RPM 仓库。

```
# rpm -ivh http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
```

接着添加 Puppetlabs 的 GPG 密钥。GPG 密钥的作用是用来验证下载软件包的完整性以及与官方网站软件包的一致性。

```
# rpm -import http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
```

最后通过 yum 方式来安装 Dashboard。

```
# yum install puppet-dashboard
```

同样的方式也适用于 Ubuntu 和 Debain 发行版本，通过 apt-get 方式安装 Dashboard。

首先编辑 /etc/apt/sources.list，添加以下内容：

```
deb http://apt.puppetlabs.com/ubuntu lucid main
deb-src http://apt.puppetlabs.com/ubuntu lucid mani
```

接着添加 Puppetlabs 的 GPG 密钥。

```
# gpg -recv-key 4BD6EC30
# gpg -a -export 4BD6EC30 > /tmp/key
# apt-key add /tmp/key
```

然后更新 APT。

```
# apt-get update
```

最后安装 Dashboard。

```
# apt-get install puppet-dashboard
```

2）通过源码方式安装 Dashboard。源码方式通常适用于企业内部网无法访问互联网的场景。首先下载 Dashboard-1.2.23 的软件包。

```
# wget http://downloads.puppetlabs.com/dashboard/puppet-dashboard-1.2.23.tar.gz
```

下载后将 puppet-dashboard-1.2.23.tar.gz 解压到程序发布的目录，通常为 Web 程序发布目录，以 Apache 为例发布目录在 /var/www/html 下。

```
# tar -xvzf puppet-dashboard-1.2.23.tar.gz -C /var/www/html
```

3）通过 GIT 克隆方式安装 Dashboard 的源码。适用于各种操作系统发行版本。

```
# git clone git::/github.com/puppetlabs/puppet-dashboard.git
# cd puppet-dashboard
# git checkout v1.2.23
```

4. 升级 Dashboard

如果我们使用的 Dashboard 版本比较老，可以通过以下方式对 Dashboard 进行最新版本的升级。

```
# yum update dashboard
```

13.2 配置 Dashboard

Dashboard 是一款 Web 程序，它的展示与配置需要数据库的支持。在初始化 Dashboard 环境后，需要通过以下 4 步来配置 Dashboard 与数据库连接的设置。

步骤 1 配置 MySQL 环境。

步骤 2 编辑 Dashboard 的 YAML 配置文件。

步骤 3 通过 Rack 填充数据库。

步骤 4 运行 Dashboard。

1. 配置 MySQL 环境

通常 MySQL 数据库使用的是默认的配置，并不适合所有的线上系统运行环境，在使用前我们需要根据业务的场景对 MySQL 参数进行优化，让它的性能发挥到极致。使用 Dashboard 也一样，修改 MySQL 配置（默认位置 /etc/my.cnf）文件中的 max_allowed_package 参数，MySQL 会根据此默认配置参数限制 Server 接收的数据包大小。某些情况下，在插入或更新 MySQL 中的数据时会被 max_allowed_packet 参数限制而导致失败，所以建议修改此值，根据硬件的配置推荐设置 32M 或者更大。修改方式分为两种，具体如下：

```
# 方式 1：修改 /etc/my.cnf 中的 max_allowed_package 参数，将此值改为 32M 或者更大，修改后重启 MySQL 守护进程
max_allowed_packet = 32M
# 方式 2：通过 MySQL 的 client 来直接修改，修改后即生效（MySQL 重启后会失效）
Mysql > set max_allowed_packet= 33554432
```

按方式 1 修改后，重新启动 MySQL 数据库，加载最新的配置。

```
# service mysqld restart
```

接着连接 MySQL 数据库，创建 Dashboard 所需的用户和库，并授权。

```
# 打开 MySQL
# mysql -uroot（账户名）-p（密码）
# 创建 dashboard 库，并设置 utf-8 字符集
mysql >  create database dashboard character set utf8;
# 创建 dashboard 用户，并设置用户密码为 dashboard
mysql >  create user 'dashboard'@'localhost' IDENTIFIED BY 'dashboard';
```

```
# 对 dashboard 用户，在本地 (localhost) 访问 MySQL 进行授权
mysql >  grant all privileges on dashboard.* to 'dashboard'@'%';
# 刷新配置
mysql > flush privileges;
```

2. 编辑 Dashboard 的 YAML 配置文件

Dashboard 的配置文件通常存放在 /usr/share/puppet-dashboard/config/ 目录中。主要是修改 databases.yml 和 settings.yml 文件。下面将分别介绍这两个文件配置文件。

（1）databases.yml 配置文件

databases.yml 配置文件主要用于 Dashboard 连接 MySQL 的参数设置，与 Puppet 的“环境”一样，它包含了 3 种环境的配置信息，分别是 production（线上环境）、developemnt（开发环境）和 test(测试环境)，每个环境中都包含了独立的连接 MySQL 的账号、密码、库、字符集和环境等信息。而这些配置信息就存放在 databases.yml 配置文件中。databases.yml 配置文件为 YAML 格式，文件内容如下：

```
#  开始
production:     # 环境
  database:     # 数据库名
  username:     # 访问数据库的用户名
  password:     # 访问数据库的密码
  encoding:     # 访问数据库的字符集
  adapter:      # 访问数据库的连接方式
development:
  省略
test:
  省略
#  结束
```

如以上配置所示，由于 development 和 test 环境的设置内容与 production 一致，所以这里作了省略。我们以 production（线上环境）为例，database 参数设置访问的库名；username 参数设置访问 MySQL 的用户名；password 参数设置访问 MySQL 的账户密码；encoding 设置连接的字符集；adapter 设置访问的方式。根据之前的 MySQL 数据库的配置，我们设置 production（线上环境）参数为以下内容（其他环境读者可以在 MySQL 创建相应的库、表、账号和密码后分别设置，由于配置方法一致，这里不再介绍）。

```
production:
  database: dashboard      # 访问数据库名
  username: dashboard      # 访问数据库账号
  password: dashboard      # 访问数据库密码
  encoding: utf8           # 数据库连接字符集
  adapter: mysql           # 访问数据库驱动
```

（2）settings.yml 配置文件

settings.yml 用于 Dashboard 的环境配置。配置文件中包含时区设置、日志上报格式设置和自定义上报地址设置和 Web 页面的翻页数等设置。settings.yml 配置文件常用参数如表 13-1 所示。

表 13-1 settings.yml 文件常用参数设置

参数	含 义
datetime_format	时间格式，默认为 '%Y-%m-%d %H:%M %Z
custom_logo_url	logo 设置
time_zone	时区设置。推荐设置 Asia/Shanghai，即中国时区
nodes_per_page	nodes 分页设置
classes_per_page	classes 分页设置
groups_per_page	group 分页设置
reports_per_page	reports 分页设置

3. 通过 Rack 填充数据库

Rack 是 Ruby on rails 程序的命令接口，通过它可以将 Dashboard 默认的库与表结构导入 MySQL 数据库中。在配置好 databases.yml 文件后，只需要通过一条命令即可填充 Dashboard 的数据库。在填充数据前需要确定自己的位置是否在 puppet-dashboard 根目录中，即 /usr/share/puppet-dashboard 下。如果不在，需要通过 cd 命令切换到 puppet-dashboard 根目录中。切换与填充命令如下：

```
# cd /usr/share/puppet-dashboard
# sudo -u puppet-dashboard rake db:migrate RAILS_ENV=production
```

Rails 本身不包含运行过程中的“环境”，需要通过 RAILS_ENV = production 设置 Rails 的工作环境变量。每次运行时都需要制定该环境变量，同时也可以通过此环境变量来切换不同的工作环境，如 developemnt（开发环境）和 test（测试环境）等。

填充命令执行后，再次通过 mysql 命令来查看填充的结果。以下为通过 Rack 成功填充后的结果。

```
# mysql -u dashboard -p dashboard  -D dashboard -e "show tables;"
+------------------------------+
| Tables_in_dashboard          |
+------------------------------+
| delayed_job_failures         |
| delayed_jobs                 |
| metrics                      |
| node_class_memberships       |
| node_classes                 |
| node_group_class_memberships |
```

```
| node_group_edges              |
| node_group_memberships        |
| node_groups                   |
| nodes                         |
| old_reports                   |
| parameters                    |
| report_logs                   |
| reports                       |
| resource_events               |
| resource_statuses             |
| schema_migrations             |
| timeline_events               |
+-------------------------------+
```

下面来简单介绍一下以上的输出内容：

- -u 参数接 MySQL 账户名。
- -p 参数接账户密码。
- -D（大写）参数接库名。
- -e 参数接查询的 SQL 语句。

输出的表内容为 Rack 填充后结果，可以根据表的名字了解其大概的内容，这里不详细介绍。

4. 运行 Dashboard

一切配置好后，我们来运行 Dashboad。Dashboard 是一个 Ruby on Rails 的程序，所以它有多种运行方式，如通过内置的 Webrick 或者 Passenger 来运行，两者的不同之处在于 Webrick 配置简单可以直接适用，但缺点是当很多 Agent 向 Dashboard 进行同时汇报时，它的性能会比较差；而 Passenger 的优势是更适合这种多 Agent 汇报的场景，具有良好的性能，但缺点是配置比较复杂。不过为了检验我们刚刚的配置是否成功，还是优先使用默认的 Webrick 的方式来运行 Dashboard。Passenger 的运行方式将会在下一节介绍。

通过 Webrick 运行的方式非常的简单，可以通过 Dashboard 的 server 命令来进行启动。

```
# /usr/share/puppet-dashboard/script/server -e production -d
```

如以上的命令所示，其中 -e 表示启动的环境。除此参数外，server 命令还包含了其他常用的参数，如表 13-2 所示。

表 13-2 server 参数列表

参 数	含 义
-p	绑定端口，Dashboard 启动后默认会在 IP 0.0.0.0 上监听 3000 端口
-b	绑定网卡接口
-d	以守护进程方式启动
-e	包括 production（线上环境）、developemnt（开发环境）和 test（测试环境）等
-P	Rails 程序的挂载目录
-h	查看帮助手册

成功启动后，以笔者的测试环境为例，结果如图 13-1 所示。在浏览器中输入 http://IP:3000 来访问 Puppet 的 Dashboard[⊖]。

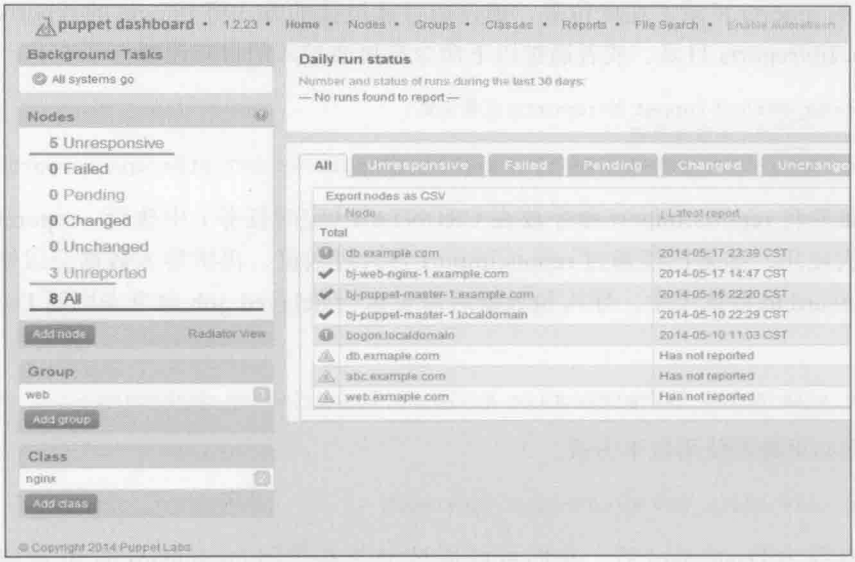


图 13-1 Dashboard 用户界面

13.3 Dashboard 应用场景

到目前为止我们已经基本配置好 Dashboard 的使用环境了。下面我们要做的是：

- 导入 Agent 的报告。
- 实时汇总 Agent 的报告。
- 使用 Dashboard 作为节点分类器。
- Dashboard 日志、数据备份与性能提升。
- Dashboard 的安全管理。

1. 导入 Agent 的报告

我们在第 12 章曾介绍过报告，Agent 将报告以文件形式上报到 Master 的指定目录中，Dashboard 可以将这些报告数据导入 MySQL 中并通过 Dashboard 程序进行沉淀与展示。报告导入的方式非常简单，切换到 Dashboard 的根目录（/usr/share/puppet-dashboard），然后通过 reports:import 命令来导入 Agent 的报告。具体如下：

⊖ 如果页面显示不正常，请确认一下浏览器的型号与发行版本，目前 Dashboard 支持的浏览器包括 Chrome、Firefox 3.5 或更高版本、Saferi 4 或更高版本和 IE 8 或更高版本。

```
# sudo rake RAILS_ENV=production reports:import
```

默认情况下命令会在 /var/puppet/lib/reports 目录中查询并导入已经生成的报告文件。如果 Puppet 的 reports 目录不在此位置，建议通过软链接的方式将 Puppet 的 reports 目录指向 /var/puppet/lib/reports 目录，或者通过以下命令来更改导入的目录位置。

```
# reports_path 为 Puppet 的 reports 目录位置
# reports_path 为报告路径
# sudo rake RAILS_ENV=production reports:import REPORT_DIR=reports_path
```

通常需要将 reports:import 命令放在 CRONTAB（定时任务）中执行。reports:import 命令可以多次使用，如果已经通过 repots:import 导入过数据，再次导入数据会以增量方式追加到 Dashboard 的数据库中。导入包数据后需要运行 delayed_job 命令来刷新 Dashboard 的数据。

```
# sudo rake RAILS_ENV=production script/delayed_job -p dashboard -n 8 -m start
```

官方网站更推荐使用以下方式。

```
# sudo rake RAILS_ENV=production jobs:work &
```

将报告导入 Dashboard 后，我们可以通过浏览器访问 Dashboard 的 IP 看到结果，如 13-2 所示。

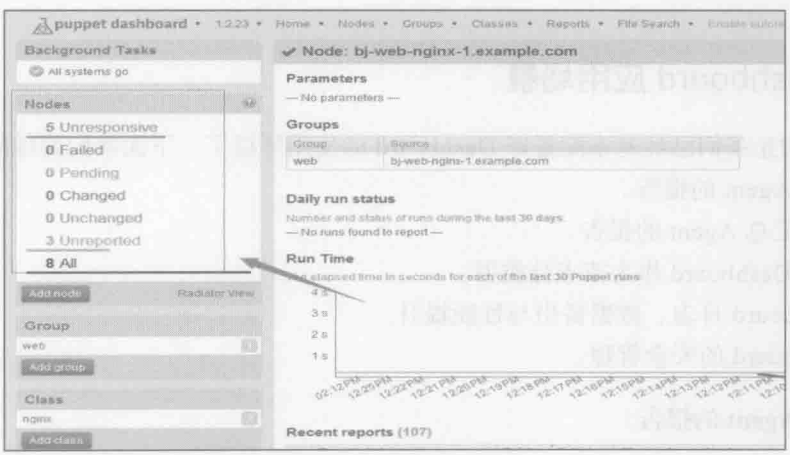


图 13-2 Dashboard 的报告展示

在图 13-2 所示 Puppet Dashboard 报告的左侧可以看到 Puppet 报告的计数器。计数器中包含了 7 种上报状态，它们分别的作用如下。

1）Unresponsive：无响应，由于 Agent 自身的原因，没有向 Puppet 汇报本身的状态。Master 如果遇见此状态，默认 1 小时内不会再处理 Agent 的上报请求，我们也可以修改 settings.yml 中的 no_longer_reporting_cutoff 来更改默认时间。

2）Pending：报告等待处理状态。报告已经到达 Master，但处理进程 delayed job 已经在运行中。

3）Failed：Agent 最后运行失败的状态。

4）Changed：Agent 最后运行状态成功，并通过 catalog 变更了服务器上的状态。

5）Unchanged：Agent 最近一次上报成功，但是 Catalog 中并没有任何变更信息。

6）Unreported：Agent 没有上报报告。

7）All：所有类别上报报告的计数器。

2. 实时汇总 Agent 报告

Dashboard 支持实时将 Agent 上报的报告进行汇总展示。需要注意的是，在 Puppet 2.6.* 版本中需要开启报告上报的开关 report = true，开启开关修改 /etc/puppet/puppet.conf 文件，追加以下内容到配置文件中。

```
[agent]
report = true
```

Puppet 2.7.* 或 Puppet 3 版本下此功能是自动开启的。实时汇总 Agent 报告，其实是将 Agent 的报告通过 HTTP 协议实时上传到 Dashboard 服务器的指定目录，再通过 Dashboard 进行汇总与展示。开启实时汇总 Agent 报告功能，编辑 Master 的 /etc/puppet/puppet.conf 配置文件，追加以下内容到配置文件中。

```
[master]
reports = store,http
reporturl = http://dashboard.example.com:3000/reports/upload
```

如以上配置文件所示，在第 11 章我们曾介绍 reports 参数为开启上报的参数，其中 store 表示将报告存放在本地磁盘；http 表示将报告以 HTTP 协议发送到指定端口，默认会发送到本机的 3000 端口（Dashboard 默认的端口）。reporturl 不是必需的参数，如果要更改汇总报告的地址或端口可以通过此参数进行修改。更改 puppet.conf 配置文件后重启 Master 的守护进程，配置才会生效。

3. 外部节点分类器

Puppet 除了对一些报告进行展示外，还可以与 ENC 结合来实现 Web 化管理 Agent。我们曾在第 10 章介绍过 ENC，ENC 比较灵活，可以适用于海量服务器的管理，而 Dashboard 正是利用了 ENC 比较灵活这一功能，通过 Rake API 来管理配置节点、类和组等功能。在通过 Dashboard 配置前，首先开启 ENC 的配置，编辑 /etc/puppet/puppet.conf 配置文件，将以下内容追加到配置文件中。

```
[master]
node_terminus = exec     # 以 exec (ENC) 方式编译 catalog
```

```
    external_nodes = /usr/bin/env PUPPET_DASHBOARD_URL=http://localhost:3000 /opt/
puppet-dashboard/bin/external_node     # 指定 ENC 脚本位置
```

在 Dashboard 中提供了 3 种类型的配置，分别是节点配置、类配置和组配置。

（1）节点配置

节点配置就是 Puppet 的节点通过 Web GUI 管理，如图 13-3 所示。我们可以通过 Web 界面手动添加节点、节点的描述与对应的 Class（类）文件，Agent 访问 Master 时会根据设置的节点，加载 Class（类）文件。

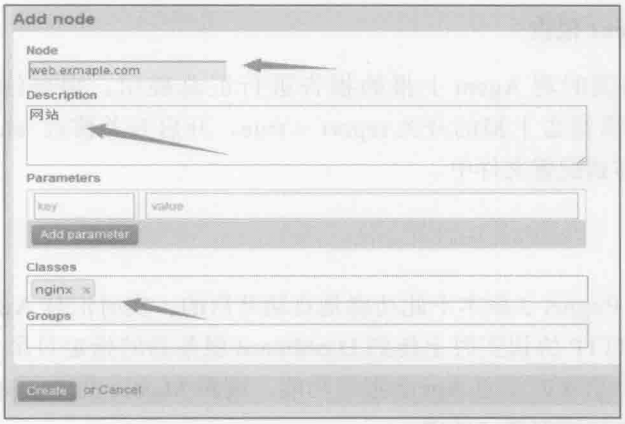


图 13-3 Puppet 节点的配置

（2）类的配置

通过单击 add class 添加类的配置，如图 13-4 所示。目前 Dashboard 只支持添加类字而不能添加类的内容，类的内容仍然需要通过手动在 Master 上编写清单文件来完成。

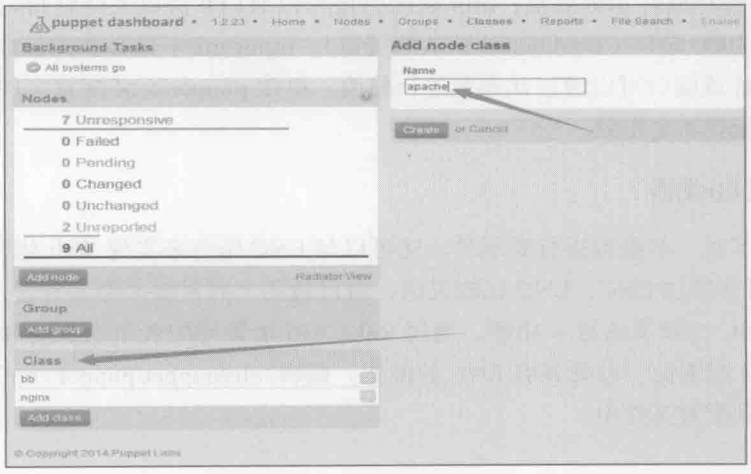


图 13-4 类的配置

（3）组的配置

组的配置是一种 Dashboard 对节点的分类方式，并没有在清单中体现，如图 13-5 所示。创建一个组后，我们可以在组内添加参数和类，这些都会叠加应用到组的所有节点上。

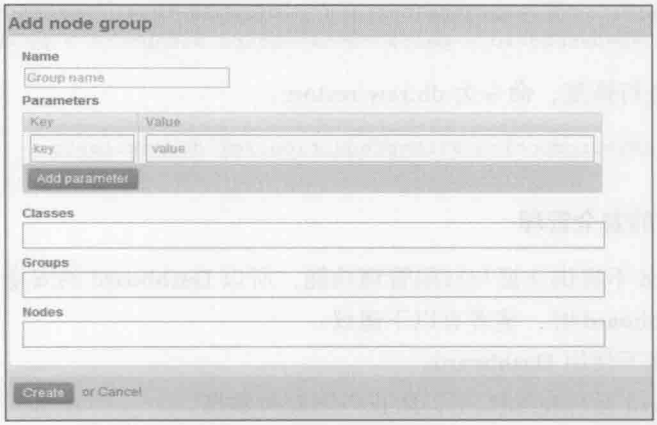


图 13-5 组的配置

4. Dashboard 日志、数据备份与性能提升

本节通过 Dashboard 日志、数据备份与性能提升来介绍 Rake 的常用命令，如表 13-3 所示。其实 Rake 命令非常强大，除了这些常用参数外，它还有大约 160 个参数，可以通过 rake –T 参数来查看它们。关于 rake 命令的更多参数可以参考 http://www.rubycc.com/column/rails3.2.3/rake.htm。

表 13-3 Rack 常用命令

命 令	作 用
log:clear	日志清理
db:raw:optimize	数据库优化
db:raw:dump	数据库备份
db:raw:restore	数据库还原

首先 Dashboard 和其他的 Rails 程序一样，会将整个的运行过程的信息记录在 /usr/share/puppet-dashboard/logs 目录中，并将日志分别存放在 production、developemnt 和 test 文件中。当 Dashboard 出现问题时，这些日志就变成了很有价值的信息，可以通过诊断日志排查 Dashboard 的故障信息。这些日志会随着访问量不断地增长，我们需要定时对此目录中的文件进行清理，以避免日志将磁盘充满，影响其他服务的正常运行。这里 Rake 本身就提供了清理命令，具体如下：

```
# cd /usr/share/puppet-dashboard; rake log:clear
```

不管我们通过哪种方式来清理日志，最好将它写入系统的 CRONTAB（定时任务）中，来定时清理残留日志。

Dashboard 和很多数据库系统一样，需要通过对 MySQL 的优化来提升它的性能，除此之外它还提供了对数据库的备份与恢复功能。

1）优化数据库。Rake 本身自带了对 MySQL 优化的命令 db:raw:optimize。

```
# rake RAILS_ENV = production db:raw:optimize
```

2）对数据库进行备份，命令为 db:raw:dump。

```
# rake RAILS_ENV=production db:raw:dump    数据库备份
mysqldump --add-locks --create-options --disable-keys --extended-insert --quick
--set-charset --user=dashboard --password=dashboard dashboard > production.sql.tmp
```

3）对数据库进行恢复，命令为 db:raw:restore。

```
# rake RAILS_ENV=production FILE=production.sql db:raw:restore
```

5. Dashboard 的安全管理

默认 Dashboard 不提供认证与权限管理功能，所以 Dashboard 的安全一直是人们关注的问题。在使用 Dashboard 时，笔者有以下建议：

1）在内网环境下使用 Dashboard。

2）将 Dashboard 运行在 SSL 下，防止内网监听数据。

3）开启防火墙限制（iptables），限制来源 IP 访问。

4）通过 HTTP Basic Authentication 进行账户认证。以下为 Apache 的认证的配置（作为参考）。

```
<Location "/">
    Order allow deny
    Allow from 192.168.240.110    # IP 限制，只准许 Puppet Master 访问
    Satisfy any
    AuthName "Puppet Dashboard"    # 认证账户名
    AuthType Basic
    AuthUserFile /etc/apache2/htpasswd    # 指定账户的密码文件
    Require valid-user
</Location>
```

13.4 Dashboard 与 Nginx 提升性能

本节主要介绍如何通过 Dashboard 与 Nginx+Passenger 结合提升 Web 的处理性能。由于 Puppet 自带的 Webrick 性能不是很强，在管理的服务器比较多而服务器同时上报时就会触发 Webrick 的处理瓶颈，影响正常的请求，所以我们要替换 Webrick，提升服务器的处理性能。如果将 Dashboard 应用在生产环境，推荐通过 Nginx+Pssenger 的方式替换 Webrick 来运行 Dashboard。

Nginx+Pssenger 曾在第 11 章介绍过，这里不重复介绍安装过程，读者可以参考第 11 章有关 Passenger 的内容。这里我们只关注 nginx.conf 配置文件的内容，将以下配置追加到 nginx.conf 配置文件的尾部。

```
server {
  listen 3000;
  server_name  dashboard.example.com;
  passenger_enabled  on;
  passenger_set_cgi_param  HTTP_X_CLIENT_DN $ssl_client_s_dn;
  passenger_set_cgi_param  HTTP_X_CLIENT_VERIFY $ssl_client_verify;
  root /usr/share/puppet-dashboard/public;
 }
```

如以上配置所示，listen 3000 为 Dashboard 的默认监听端口；server_name 参数指定 Dashboard 的域名地址；passenger_enabled 开启 passenger 功能；passenger_set_cgi_param 参数设置 HTTP 头信息，HTTP 头信息主要为 Agent 证书访问提供认证的功能；root 参数指定 Dashboard 的发布目录。修改 Nginx 的配置文件后，不要忘记通过 server nginx reload 重新加载 Nginx 的配置文件，然后在浏览器中输入 http://dashboard.example.com:3000，如图 13-6 所示。

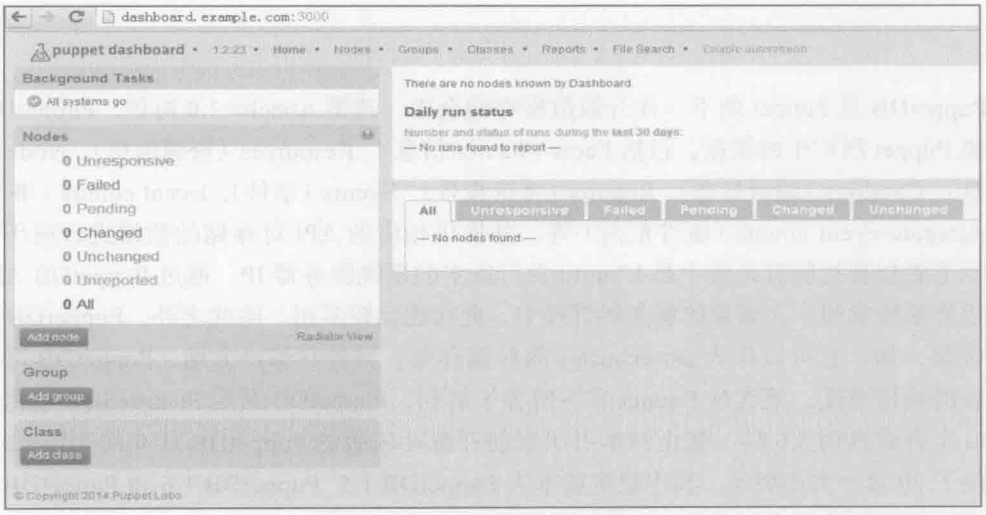
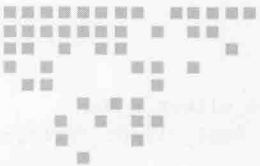


图 13-6 Dashboard 与 Nginx+Passenger



Chapter 14 第 14 章

PuppetDB 数据仓库

PuppetDB 是 Puppet 的下一代开源数据存储仓库，遵循 Apache 2.0 协议，PuppetDB 可以收集 Puppet 所产生的数据，包括 Facts（Facter 信息）、Resources（资源信息）、Nodes（节点信息）、Catalogs（编译日志）、Reports（系统报告）、Events（事件）、Event counts（事件数）和 Aggregate event counts（事件汇总）等，并提供相应的 API 对存储的数据进行遍历与检索。如笔者想查找所有系统中是 Ubuntu 发行版本的系统服务器 IP，通过 PuppetDB API 就可以很容易检索到。在海量的服务器管理中，此功能比较实用。除此之外，PuppetDB 还有很多优势，如，它可以作为 storeconfigs 的后端存储。另外在编译方面，PuppetDB 可以加快数据的编译速度，笔者从 Puppet 官方博客了解到，PuppetDB 利用 PostgreSQL 数据库编译 650 个资源只用 5.6 秒。截止到本书出版前开源社区版的 PuppetDB 从 0.10.0 到 2.0.0 已经历经了 20 多个大小版本，其中稳定版本为 PuppetDB 1.5、PuppetDB 1.6 和 PuppetDB 2.0。关于 PuppetDB 版本之间的差异请参考官方网站 http://docs.puppetlabs.com/puppetdb/2.0/release_notes.html。本章以 PuppetDB 2.0 版本为例，首先介绍 PuppetDB 环境的安装；接着介绍 PuppetDB 的配置；然后介绍 PuppetDB API；最后是有关 PuppetDB 的问答。

14.1 PuppetDB 环境安装

本节通过 CentOS 6.5_x86-64 位系统来安装开源社区的 PuppetDB 2.0 版本。与很多软件安装方式一样，安装前需要注意以下事项：

1）Puppet 企业版本安装时默认自带 PuppetDB，所以无须再次安装，而 Puppet 开源社区版本需要自行安装 PuppetDB。

2）PuppetDB 本身并不能存储数据，它利用自带的 HSQLDB 或 PostgreSQL 数据库对其数据进行存储。HSQLDB 是 PuppetDB 默认的数据库，所以不用单独安装。如果要使用 PostgreSQL 数据库，需要预先安装它的环境（PuppetDB 不支持 MySQL 数据库，因为 MySQL 缺乏对递归查询的支持）。

3）PuppetDB 的不同版本对 Facter 与 Puppet 版本的要求也不一致，PuppetDB 2.0 版本需要安装 Facter1.7.0 或者更高版本。对于 Puppet 版本，早期的 PuppetDB 0.9 到 PuppetDB 1.6 只支持 Puppet 2.7.12 或者更高版本，而 PuppetDB 2.0 需要安装 Puppet 3.5.1 或者更高的版本，所以在安装 Puppet 环境时需要注意版本的要求。

PuppetDB 目前只支持 UNIX/Linux 系列操作系统，UNIX/Linux 系列操作系统发行版本如下：

- RedHat 企业版 Linux 5 或者 6，也包括 CentOS 5 或者 6。
- Debian。
- Ubuntu 12.10、12.04 LTS、10.04 LTS。
- Fedora 18、19 和 20。

14.1.1 PuppetDB 辅助环境安装

PuppetDB 需要 JDK（Java Development Kit）的支持，JDK 是 Java 语言的软件开发工具包，PuppetDB 2.0 需要安装 JDK1.70 或以上版本，我们通过 JAVA 包来安装 JDK 的环境。

```
# yum install java
```

目前 PuppetDB 后端可以使用两种数据库，一种是默认的 HSQLDB 数据库，另外一种是 PostgreSQL 数据库。

HSQLDB 是 PuppetDB 自带默认的数据库，它是纯 Java 开发的数据库，可以通过 JDBC Driver 来存取数据，支持 ANSI-92 标准的 SQL 语法，而它所占用的空间很小，大约只有 160KB，并且拥有快速的数据库引擎功能。另外 HSQLDB 数据库也为我们提供了一些辅助工具，如 WEB-SEVER、缓冲查询及一些管理工具等。HSQLDB 数据库属于 BSD 的 license，可以自由下载使用，并且可以安装使用在商业产品上。

PostgreSQL 数据库是加州大学伯克利分校计算机系开发的基础的对象关系型数据库管理系统（ORDBMS）。PostgreSQL 数据库支持大部分 SQL 标准，并且提供了许多其他现代特性：复杂查询、外键、触发器、视图、事务完整性和 MVCC 等。同样，PostgreSQL 数据库可以用许多方法扩展，如增加新的数据类型、函数、操作符、聚集函数、索引免费使用、修改和分发等。

对于两种数据库 Puppet 官方网站更推荐使用 PostgreSQL 数据库，所以本节对 PostgreSQL 9.4 数据库版本进行安装演示，以下为 PostgreSQL 数据库安装方式。

1）默认情况下系统源并不包括 PostgreSQL 数据库，需要手动安装源。以下为 PostgreSQL

9.4 数据库源安装的方法。

```
# yum install http://yum.postgresql.org/9.4/redhat/rhel-6-x86_64/pgdg-centos94-9.4-1.noarch.rpm
```

2）PostgreSQL 数据库的安装方法如下：

```
# 数据库的安装
# yum install postgresql*
```

14.1.2 PuppetDB 环境安装与升级

1. PuppetDB 安装

PuppetDB 安装的成本相对来说比较低，官方网站提供了多种的安装方式。下面来介绍常见的两种 PuppetDB 2.0 安装方式。

方式 1 首先来看通过 yum 源安装 PuppetDB，这是最简单的方式。可以根据操作系统发行版本来选择不同的源，Puppet 官方源地址为 http://yum.puppetlabs.com。安装过程如下。

1）安装 Puppet 官方源。

```
# yum install http://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
```

2）PuppetDB 环境安装（如果不指定 PuppetDB 的安装版本，默认为最后一个版本，即 PuppetDB 2.0）。具体方法如下：

```
# yum install puppetdb puppetdb-terminus
```

另外 Puppet 官方还推荐通过 Puppet 提供的命令来安装，通过 Puppet 安装的优势是可以不区分操作系统的发行版本。具体如下：

```
# puppet resource package puppetdb ensure=latest
# puppet resource package puppetdb-terminus ensure=latest
```

3）PuppetDB 在启动前需要配置证书文件，为后续的 SSL 做准备，SSL 的优势是防止数据在网络传输过程中被截获与监听，SSL 的作用是提高数据的安全性。PuppetDB 可以与 Puppet 共用证书文件，所以如果在已经安装好的 PuppetDB 服务器上同时运行着 Puppet 的守护进程，可以通过以下命令让 PuppetDB 与 Puppet 共同使用一张证书文件。

```
# /usr/sbin/puppetdb ssl-setup
```

以上命令的作用是查找本机的 Puppet 证书文件，并将证书文件复制到 PuppetDB 证书文件所在的 /etc/puppetdb/ssl 目录中。如果 PuppetDB 与 Puppet 不在同一台服务器上工作，需要通过以下方式将 Puppet 证书文件远程复制到 PuppetDB 所在服务器的 SSL 目录。

```
# scp $(puppet master --configprint ssldir)/ca/ca_crt.pem puppetdb.example.com:/
```

```
etc/puppetdb/ssl/ca.pem
    # scp $(puppet master --configprint ssldir)/private_keys/puppetdb.example.com.pem
puppetdb.example.com:/etc/puppetdb/ssl/private.pem
    # scp $(puppet master --configprint ssldir)/certs/puppetdb.example.com.pem
puppetdb.example.com:/etc/puppetdb/ssl/public.pem
```

如以上命令所示，通过系统 scp 命令将本机文件复制到远程服务器，其中通过 puppet master --configprint ssldir 命令来打印证书所在目录位置路径，后接证书文件地址 /ca/ca_crt.pem，puppetdb.example.com 为远程 PuppetDB 的域名，“：”后边所接的为 PuppetDB 服务器同步配置文件的目标路径。同步证书文件后需要授权证书文件访问权限。具体如下：

```
# chown puppetdb:puppetdb /etc/puppetdb/ssl/*.pem
# chmod 0600 /etc/puppetdb/ssl/*.pem
```

方式 2 通过源码方式安装 PuppetDB。相对于方式 1 来说方式 2 要更加的复杂，但优势是更加的通用。以下为安装过程：

1）安装 Leininge（Leiningen 是一个 Clojure 项目管理工具，可以让开发者轻松地将 clojure 类库发布到 Clojars 上）。具体方法如下：

```
# mkdir ~/bin && cd ~/bin
# curl 'https://raw.github.com/technomancy/leiningen/stable/bin/lein' -o lein
# chmod ugo+x lein
# ./lein
# symlink lein to somewhere in your $PATH
# sudo ln -s /full/path/to/bin/lein /usr/local/bin
```

2）通过 GIT 复制 PuppetDB 源码到本地，并通过命令初始化 PuppetDB 环境。

```
# mkdir -p ~/git && cd ~/git
# git clone git://github.com/puppetlabs/puppetdb
# cd puppetdb
# rake package:bootstrap
# sudo LEIN_ROOT=true rake install
```

源码安装方式会将 PuppetDB 初始化脚本安装到 /etc/init.d/ 目录中，只需要创建 mkdir –p /etc/puppetdb 配置文件目录即可。

3）重复方式 1 中的第三步。

2. PuppetDB 升级

对 PuppetDB 进行升级比较简单，如果读者仍然使用老的 PuppetDB 版本，笔者建议升级到 PuppetDB 2.0 版本。升级分为如下 3 步：

1）停止 PuppetDB 守护进程。

```
# puppet resource service puppetdb ensure=stopped
```

2）升级 PuppetDB 到 2.0 版本（ensure= 后接版本，lastest 表示最后一个版本，目前 PuppetDB 最后一个版本为 PuppetDB 2.0）。

```
# puppet resource package puppetdb ensure=latest
```

3）重新启动 PuppetDB 守护进程。

```
# puppet resource service puppetdb ensure=running
```

14.2 PuppetDB 与 Puppet 结合配置

Agent 可以通过 Puppet 与 PuppetDB 的结合，将 Puppet 的资源、编译信息和 Facter 等信息导入 PostgreSQL 数据库，同时也可以通过 Puppet API 方便地检索到这些导入 PostgreSQL 数据库中的信息，如图 14-1 所示。所以本节会按照图 14-1 所示的流程倒序介绍数据库、PuppetDB 和 Puppet 的配置。

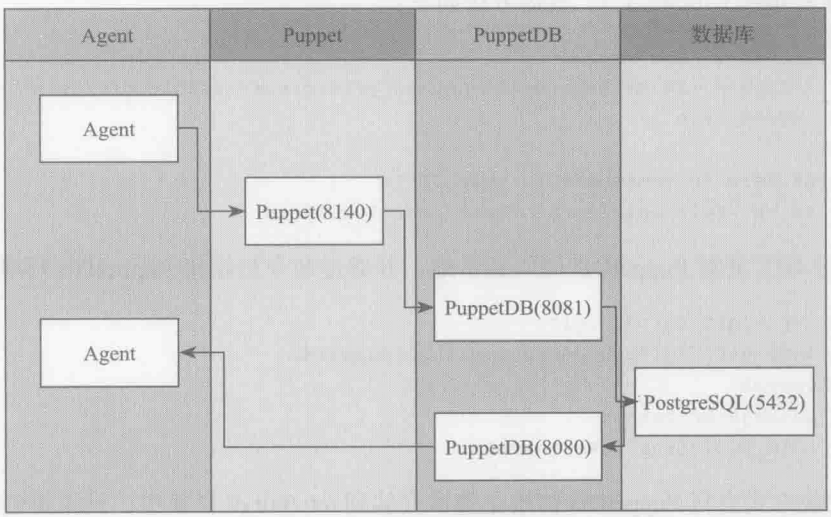


图 14-1 Agent 导入与导出数据流程图

14.2.1 数据库配置

Puppet 官方推荐 PostgreSQL 数据库作为 PupptDB 后端存储，所以本节主要介绍 PostgreSQL 数据库的配置。由于 PostgreSQL 数据库的守护进程监听在 IP127.0.0.1 上，而 PostgreSQL 配置文件写的是 localhost 的域名，所以我们需要预先指定 localhost 域名的 IP 为 127.0.0.1，以防止出现连接 PostgreSQL 数据库失败的情况。编辑 /etc/hosts 文件并追加以下内容。

```
# /etc/hosts
127.0.0.1 localhost localhost.localdomain
```

接着修改 PostgreSQL 数据库的 /var/lib/pgsql/9.4/data/pg_hba.conf 文件，追加以下访问授权信息。

```
# vim /var/lib/pgsql/9.4/data/pg_hba.conf
# IPv6 local connections:
host        all     all     ::1/128                          ident   # 改为 ident
# "local" is for Unix domain socket connections only
local       all     all                                      trust   # 改为 trust
# IPv4 local connections:
host        all     all     127.0.0.1/32                     trust   # 改为 trust
```

然后启动 PostgreSQL 数据库的守护进程。启动方式如下：

```
# /etc/init.d/postgresql-9.4 start
```

启动 PostgreSQL 数据库后，建议通过 nentstat –tnl 命令查看守护进程监听端口是否存在，如果存在则说明已经启动成功。

```
# netstat -tnl | grep 5432
tcp     0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
```

注意 如果 PostgreSQL 数据库启动失败，可以通过 /var/lib/pgsql/9.4/pgstartup.log 日志定位错误的原因。

接着在 PostgreSQL 数据库配置好的基础上，创建 PuppetDB 的库、访问账户与密码。

```
# su -u postgres sh     # PostgreSQL 不支持 root 账户直接访问，需要转为 postgres 账户
$ createuser -DRSP puppetdb     # 创建账户与密码
$ createdb -E UTF8 -O puppetdb puppetdb     # 创建账户对应的库
$ exit
```

通过账户与密码访问 PostgreSQL 数据库，测试刚刚创建的账号与密码是否正确。

```
# -h 访问域名，-U（大写）访问用户名，-d 访问数据库名，-W 访问数据库密码
# psql -h localhost -U puppetdb -d puppetdb -W
```

成功连接 PostgreSQL 数据库后，通过数据库的 psql 命令，输入“\l”（此命令与 MySQL 的 show databases 命令功能一致）查看数据库结构，如图 14-2 所示。确认 PuppetDB 库已经成功创建。

```
puppetdb=> \l
                                  List of databases
   Name    |  Owner   | Encoding |  Collation  |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 puppetdb  | puppetdb | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
           |          |          |             |             : postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
           |          |          |             |             : postgres=CTc/postgres
(4 rows)
```

图 14-2 PostgreSQL 数据库的库结构

14.2.2 PuppetDB 配置

本节首先介绍 PuppetDB 的配置文

件，并非 PuppetDB 所有的配置文件都需要修改，读者只需要关注每个配置文件的用途，并重点关注 database.ini 和 jetty.ini 两个配置文件。接着介绍 PuppetDB 启动过程，最后介绍 PuppetDB 的 Dashboard。关于 PuppeDB 配置的更多信息请参考官方网站 http://docs.puppetlabs.com/puppetdb/2.0/。

1. PuppetDB 配置

PuppetDB 的配置文件主要分为两类，第一类是初始化脚本，它包含了 PuppetDB 相关配置的环境变量与启动的守护进程所占的系统资源等，如果我们的机器配置比较高，建议通过初始化脚本中的参数进行调优，提升 PuppetDB 的性能；反之则无须更改初始化脚本配置。第二类是配置文件类，它包含了 PuppetDB 后端数据库的配置与本身监听端口等配置。两类文件中需要重点关注后者配置文件，下面来介绍这两类文件。

（1）PuppetDB 初始化脚本类

PuppetDB 安装后不同的系统发行版本初始化脚本放置的位置也不一致，如表 14-1 所示。

表 14-1 PuppetDB 初始化脚本在不同发行版本中的位置

操作系统发行版本	文 件 位 置
RedHat 与 CentOS 发行版本	/etc/sysconfig/puppetdb
Debian 和 Ubuntu	/etc/default/puppetdb
PE 企业版本相关	/etc{sysconfig，default}/pe-puppetdb

以 CentOS 系统为例，初始化脚本位置在系统的 /etc/sysconfig/puppetdb 目录下。初始化脚本中包含以下的信息。

- JAVA_BIN：Java 二进制文件位置。
- JAVA_ARGS：Java 二进制命令选项，通过 -Xmx 配置 Java 最大堆的大小。如果系统内存比较大可以修改 -Xmx 参数增加更多的内存。

```
JAVA_ARGS="-Xmx192m"     # 默认为 192M
JAVA_ARGS="-Xmx1g"       # 修改后为 1G
```

- USER：PuppetDB 守护进程运行的账户。
- INSTALL_DIR：PuppetDB 安装目录。
- CONFIG：指定 PuppetDB 配置文件的路径（/etc/puppetdb/conf.d）。

（2）PuppetDB 的配置文件类

可以在 /etc/puppetdb/conf.d 中找到。PuppetDB 主要的配置文件如下。

- config.ini：定义 [global] 全局配置和 [command-processing] 配置命令处理系统。
- database.ini：定义 [database] 数据库相关配置信息，如选择数据库类型、连接账号密码等。
- jetty.ini：定义 PuppetDB 的监听 IP、端口和证书配置文件位置等。
- repl.ini：定义 [repl] 远程运行修改配置。

需要重点关注 database.ini 和 jetty.ini 两个配置文件。首先介绍 database.ini 配置文件，

刚刚已经介绍过它的作用是用来指定 PuppetDB 连接的数据库。下面以配置 PostgreSQL 数据库为例，编辑 /etc/puppetdb/conf.d/database.ini 文件。内容如下：

```
[database]
# 数据库驱动
classname = org.postgresql.Driver
# 系统默认为 hsqldb 数据库，通过 subprotocol 修改默认数据库为 postgresql
subprotocol = postgresql
# 连接数据库的地址，PostgreSQL 格式为 //host:port/databaseName
subname = //localhost:5432/puppetdb
# 访问数据库账户名
username = puppetdb
# 访问数据库密码
password = puppetdb
# 数据库压缩频率，默认 60 分钟
gc-interval = 60
# 慢查询日志，单位（秒）
log-slow-statements = 10
```

下面简单分析一下上面的参数与值。

- classsname 参数：嵌入式数据库驱动，可选 org.postgresql.Driver 值或者 org.hsqldb.jdbcDriver 值，默认为 org.hsqldb.jdbcDriver 值。最终值由我们选择的数据库决定。
- subprotocol 参数：嵌入式数据库协议，即选择哪个数据库作为存储。可选 hsqldb 值或者 postgresql 值，默认为 hsqldb 值。
- subname 参数：连接数据库方式，postgresql 配置格式 //host:port/databaseName。hsqldb 配置格式为 file:/var/lib/puppetdb/db/db;hsqldb.tx=mvcc;sql.syntax_pgs=true。
- username 参数：数据库的账户名。
- password 参数：数据库的密码。
- gc-interval 参数：数据压缩频率。
- log-show-statements 参数：慢日志查询，可以通过此参数对不合理的 SQL 进行优化与改进。

接着来看 jetty.ini 配置文件，它的作用是配置 PuppetDB 守护进程的相关参数。编辑 /etc/puppetdb/conf.d/jetty.ini 文件，内容如下：

```
[jetty]
# 设置 PuppetDB 连接主机名或 IP
host = puppetdb.example.com
# 设置 PuppetDB 监听端口
port = 8080
# 设置 PuppetDB 的 SSL 监听主机或 IP
ssl-host = puppetdb.example.com
# 设置 PuppetDB 的 SSL 监听端口
ssl-port = 8081
# 私钥路径，建议 PuppetDB 私钥与 Puppet 的私钥共用
```

```
ssl-key = /etc/puppetdb/ssl/private.pem
# 公钥路径，建议 PuppetDB 公钥与 Puppet 的公钥共用
ssl-cert = /etc/puppetdb/ssl/public.pem
# CA 证书路径，建议 PuppetDBCA 证书与 Puppet 的 CA 证书共用
ssl-ca-cert = /etc/puppetdb/ssl/ca.pem
```

下面简单分析一下上面的参数与值。

- host 参数：设置监听连接的主机名或者 IP。
- port 参数：设置连接的端口。
- ssl-host 参数：设置 SSL 的监听主机名或 IP。
- ssl-key 参数：设置私钥的位置。
- ssl-cert 参数：设置私钥的位置。
- ssl-ca-cert 参数：设置 CA 根证书位置。

2. PuppetDB 启动

配置好 database.ini 和 jetty.ini 两个文件后，就可以启动 PuppetDB 守护进程了。启动方式如下：

```
# /etc/init.d/puppetdb start
```

PuppetDB 启动后再次访问 PostgreSQL 数据库，这时 PuppetDB 会自动在 PostgreSQL 数据库填充所用到的表，可以再次连接 PostgreSQL 数据库，确认 PuppetDB 成功配置。以下为 PostgreSQL 数据库连接的命令：

```
# psql -h localhost -U puppetdb -d puppetdb -W
```

连接到 PostgreSQL 数据库后，输入 \dt(此命令与 MySQL 的 show tables 命令功能一致)查看数据库表结构，如图 14-3 所示。可以看到 PuppetDB 已经填充了相关的表。

```
puppetdb=> \dt
                List of relations
 Schema |            Name            | Type  |  Owner
--------+----------------------------+-------+----------
 public | catalog_resources          | table | puppetdb
 public | catalogs                   | table | puppetdb
 public | certname_facts             | table | puppetdb
 public | certname_facts_metadata    | table | puppetdb
 public | certnames                  | table | puppetdb
 public | edges                      | table | puppetdb
 public | environments               | table | puppetdb
 public | latest_reports             | table | puppetdb
 public | reports                    | table | puppetdb
 public | resource_events            | table | puppetdb
 public | resource_params            | table | puppetdb
 public | resource_params_cache      | table | puppetdb
 public | schema_migrations          | table | puppetdb
(13 rows)
```

图 14-3 PostgreSQL 数据库的表结构

填充表后，表明已经成功地配置了 PuppetDB。

注意 如果 PuppetDB 启动失败，可以通过查看 /var/log/puppetdb/puppetdb.log 日志定位错误的原因。

3. PuppetDB 的 Dashboard

PuppetDB 还提供了 Web 的展示窗口，也就是 PuppetDB 的 Dashboard，它直观地展示了 PuppetDB 的运行状况，包括积压队列、命令处理进程、node 和资源信息等。启动 PuppetDB 后，可以在 IE 中输入 http://puppetdb.example.com:8080 来访问 PuppetDB 的 Dashboard，如图 14-4 所示。输入的 puppetdb.example.com 为 PuppetDB 的地址。

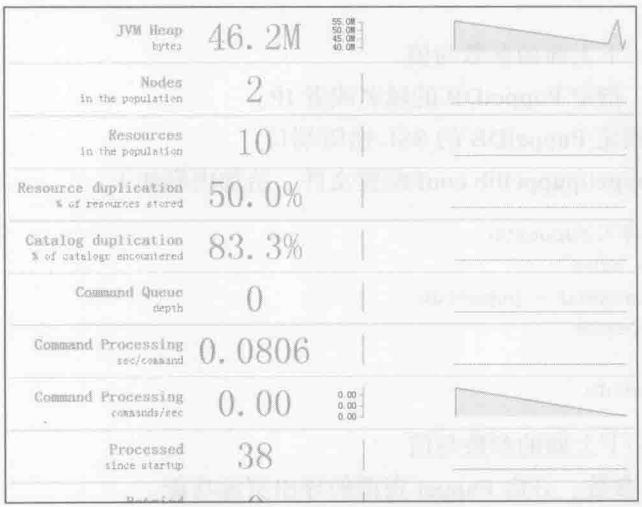


图 14-4 PuppetDB Dashboard

另外 PuppetDB 的 Dashboard 配置也比较灵活，可以通过追加参数的形式修改其外观与刷新频率。要修改的参数如下：

- width：Dashboard 的宽。
- height：Dashboard 的长。
- pollingInterval：Dasoboard 刷新的间隔，单位为毫秒。
- nHistorical：多少历史数据点使用图。

通过 http://puppetdb.example.com:8080/dashboard/index.html?height=240&pollingInterval=1000 调整它的参数，如将刷新率调整为 500 毫秒，修改参数 pollingInterval=500 即可调整 Dashboard 的刷新频率。

14.2.3 Puppet 配置

因为我们使用了 PuppetDB 2.0 版本，所以在配置 Puppet 前，首先通过 puppet –V 命令

确认 Puppet 的版本是否为 3.5.1 或者以上版本，如果不是则需要通过 yum update puppet 进行升级。确认版本没有问题后，继续配置 Puppet 与 PuppetDB。Puppet 与 PuppetDB 需要修改两个配置文件，它们分别是 puppetdb.conf 和 puppet.conf。puppetdb.conf 配置文件的作用是让 Puppet 连接 PuppetDB 的配置文件，而 puppet.conf 是 Puppet 的主配置文件。最后还要创建 routes.yaml 文件，此配置文件的作用是改变 Puppet 的默认缓存与编译的方式。

1）配置 puppetdb.conf，默认此文件并不存在，需要手动创建它。编辑 /etc/puppet/puppetdb.conf 配置文件，内容如下：

```
[main]
server = puppetdb.example.com
port =8081
```

下面简单分析一下上面的参数与值。

- server 参数：指定 PuppetDB 的域名或者 IP。
- port 参数：指定 PuppetDB 的 SSL 监听端口。

2）编辑 /etc/puppet/puppetdb.conf 配置文件，追加内容如下：

```
# 将 Puppet 数据导入 PuppetDB
storeconfigs = true
storeconfigs_backend = puppetdb
# 将报告导入 PuppetDB
report = true
reports = puppetdb
```

下面简单分析一下上面的参数与值。

- storeconfigs 参数：开启 Puppet 资源的导出资源功能。
- storeconfigs_backend 参数：指定导出后的数据存储的介质。
- report 参数：开启日志上报功能。
- reports 参数：指定报告存储介质。

3）在 Master 下创建 routes.yaml 文件。通过 puppet master --configprint route_file 命令找到 routes.yaml 文件位置，并追加以下内容到 routes.yaml 文件中。

```
---
apply:
  catalog:
    terminus: compiler
    cache: puppetdb
  resource:
    terminus: ral
    cache: puppetdb
  facts:
    terminus: facter
    cache: puppetdb_apply
```

4）配置文件增加或变更后需要重启 Puppet 的守护进程，加载更新后的配置文件。重启 Puppet 守护进程的命令如下：

```
# service puppetmaster restart
```

5）通过 Agent 来访问 Master，访问后的结果会直观地展示在 PuppetDB 的 Dashboard 上，如图 14-5 所示。

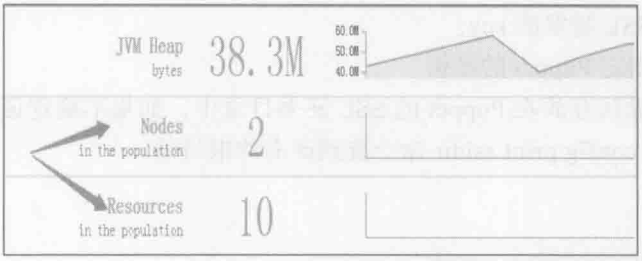


图 14-5 Puppet Dashboard 访问记录

14.3 PuppetDB API

PuppetDB API 使用了命令查询责任分离的模式（Command/Query Responsibility Separation，CQRS），它是一种架构体系模式，能够改变模型的状态的命令和模型状态的查询，从而实现分离。命令通过队列实现了异步执行，所有接收的命令会在一个消息队列中，然后命令处理子系统会采用先进先出（FIFO）原则来读取消息队列中的这些命令。可以通过 PuppetDB 的 API 对 Puppet 的数据进行检索，借助表达式与条件语句进行信息的筛选与过滤，并对筛选与过滤后的数据重新组合整理使用。本节将分两个部分来介绍 PuppetDB API，分别为 PuppetDB API 检索结构与 PuppetDB API 检索语句。

14.3.1 PuppetDB API 检索结构

首先来看一下 PuppetDB 的检索语句，PuppetDB 的检索语句的结构可以通过 HTTP 协议“GET 方式 + 查询语句”进行查询，结果以 JSON 格式返回。目前 PuppetDB 检索分为两种方式，一种方式为普通的 SSL 检索。SSL 检索的优势是比较安全，适用于安全性比较差的网络环境，如互联网，但劣势是拼写 URL 比较烦琐。SSL 检索通常适用于 Puppet 向 PuppetDB 插入或检索数据。另一种方式是直接通过 HTTP 协议查询，它的优势是拼写 URL 简单方便，但劣势是安全性比较差，更适合于内部网络。

1. SSL 检索

如要通过 SSL 方式检索 PuppetDB 中的 resources 资源类内容（SSL 检索使用 8081 端

口）。curl 就需要指定 PuppetDB 服务器的证书文件，然后才能检索 PuppetDB 中信息。

```
curl -X GET https://puppetdb.example.com:8081/v3/resources --cacert
/etc/puppet/ssl/certs/ca.pem --cert /etc/puppet/ssl/certs/thisnode.pem --key
/etc/puppet/ssl/private_keys/thisnode.pem --data-urlencode query@<filename>
```

下面简单分析一下上面的参数与值。

- cacert 参数：指定 Puppet 的 CA 证书文件。
- cert 参数：SSL 签发的 key。
- key 参数：指定 Puppet 的私钥。

私钥文件一般默认存放在 Puppet 的 SSL 证书目录中，如果不确定证书文件的位置，可以通过 sudo puppet config print ssldir 命令查到证书的根目录。

2. HTTP 检索

HTTP 检索使用 8080 端口，如下：

```
curl -X GET http://puppetdb.example.com:8080/v3/<ENDPOINT>?query=<QUERY STRING>
```

下面简要介绍一下上面检索语句的结构。

1）http://puppetdb.example.com 为 PuppetDB 的域名，8080 为服务端口。PuppetDB 通过端口来区分不同的协议，8080 端口为普通的 HTTP 协议，8081 端口为 SSL 协议。Pupppet DB 端口定义在 PuppetDB 服务器的 /etc/puppetdb/conf.d/jetty.ini 文件上。

2）V3 为 API 的版本号，截止到本书出版前 PuppetDB 共有 4 个版本，分别是 v1、v2、v3 和 v4。PuppetDB 1.3 或更高版本支持 v1、v2 和 v3 版本查询 API，而 v1 版本则结合 PuppetDB 1.0.x，向下兼容。其中 v3 为稳定版本，v4 为实验版本，v2 为即将弃用版本，所以综合各版本情况，推荐使用 v3 版本。

3）检索内容包括 Facts（Facter 信息）、Resources（资源信息）、Nodes（节点信息）、catalogs（编译日志）、Reports（系统报告）、Events（事件）、Event counts（事件数）和 Aggregate event counts（事件汇总）等。

4）检索语句支持以下内容：

- 使用 HTTP 协议，并通过 Get 方式获取数据。
- 使用 url-encoded 作编码。
- 支持 JSON 数组。
- 支持正则表达式。
- 支持条件表达式。

14.3.2 PuppetDB API 检索语句

我们可以通过 PuppetDB API 检索 Puppet 上报的信息，Puppet 上报的信息比较多，包

括 Facts（Facter 信息）、Resources（资源信息）、Nodes（节点信息）、Catalogs（编译日志）、Reports（系统报告）、Events（事件）和 Aggregate event counts（事件汇总）等。可以通过 PuppetDB API 进行信息的检索，并根据检索后的信息进行二次加工以满足自己的需求。目前 PuppetDB API 支持两种信息检索方式，一种为“URL+ 地址”方式，通过查询地址变更获取查询数据，称为“路由”方式。另一种为“URL+ 参数”方式，虽然检索方式不同，但是返回结果却一致。在介绍 PuppetDB API 检索前，先来了解一下 curl 命令，它是 Linux 下的一款很强大的 http 命令工具，通过它可以获取远程网页的信息。如果找不到 curl 命令，可以通过 yum install curl* 来安装。在 PuppetDB API 检索中大量的使用了 curl 命令进行介绍，其常用参数如表 14-2 所示。

表 14-2 curl 常用参数

参 数	含 义
-G	HTTP GET 方式
-X/--request <command>	指定什么命令
--data-urlencode	指定 HTTP 协议编码

本节将会介绍常用的 Facts（Facter 信息）、Resources（资源信息）、Nodes（节点信息）、Catalogs（编译日志）和 Reports（系统报告）类检索方式。关于 PuppeDB API 的更多信息请参考官方网站 http://docs.puppetlabs.com/puppetdb/2.0/api/。

1. Facts 类

Facts 类为 Facter 收集的数据，并会被上报到 PuppetDB 中，可以通过 PuppetDB API 检索到 Facter 上报的这些信息，并可以根据检索后的信息再次封装使用。PuppetDB API 检索 Facts 数据，目前包括 4 项返回信息，分别是 name（Facter 的键）、values（Facter 的值）、certname（节点名）和 enviroment（环境）。下面分别以路由方式与 URL 参数方式来介绍检索信息的方法。

1）Facts 类路由方式检索格式。

```
# curl -X GET http://puppetdb.example.com:8080/v3/facts     # 输出全部 Facter 值
# curl -X GET http://puppetdb.example.com:8080/v3/facts/<NAME>     # 输出 Facter 值中的 Key
# curl -X GET http://puppetdb.example.com:8080/v3/facts/<NAME>/<VALUE>     # 输出 Facter 值中的 Value
```

下面看一个路由方式检索案例。检索所有 Puppet 上报的 IP 地址对应节点的信息，具体如下。

```
# curl -X GET http://puppetdb.example.com:8080/v3/facts/ipaddress
# 以下为符合条件的返回结果
[ {
  "certname" : "bj-web-nginx-1.example.com",    # 节点名
  "environment" : "production",     # 环境
  "name" : "ipaddress",     # Facter 的键
  "value" : "192.168.110.129"     # Facter 的值
}, {
  "certname" : "localhost.localdomain",     # 节点名
```

```
    "environment" : "production",     # 环境
    "name" : "ipaddress",     # Facter 的键
    "value" : "192.168.110.130"     # Facter 的值
  }, {
    "certname" : "web.example.com",     # 节点名
    "environment" : "production",     # 环境
    "name" : "ipaddress",     # Facter 的键
    "value" : "192.168.110.130"     # Facter 的值
  }]
```

从以上输出结果中可以看到 PuppetDB API 以 JSON 格式返回数据，并显示 PuppetDB 中存储的所有 Puppet 上报的 IP 地址的相关信息。

2）Facts 类 URL 参数方式检索。命令如下：

```
curl -X GET http://puppetdb.example.com:8080/v3/facts --data-urlencode
'query=["=", "参数","值"]'
```

参数可以是 name(Facter 的键)、values(Facter 的值)、certname(节点名) 和 enviroment(环境)。如检索 name 的 key 为 operatingsystem，检索所有系统的发行版本。结果如下：

```
# curl -X GET http://puppetdb.example.com:8080/v3/facts --data-urlencode
'query=["=", "name", "operatingsystem"]'
# 以下为符合条件的返回结果
[{"certname": "a.example.com", "name": "operatingsystem", "value": "Debian"},
 {"certname": "b.example.com", "name": "operatingsystem", "value": "RedHat"},
 {"certname": "c.example.com", "name": "operatingsystem", "value": "Darwin"},
```

2. Resource（资源）类

Resource 为资源类，通过 Puppet API 可以检索数据库中已经存在的资源信息。以下为资源检索的路由格式：

```
# curl -X GET http://puppetdb.example.com:8080/v3/resource    # 输出资源信息
# curl -X GET http://puppetdb.example.com:8080/v3/<TYPE>     # 检索资源名
# curl -X GET http://puppetdb.example.com:8080/v3/<TYPE>/<TITLE> # 检索资源名与标题
```

案例 1

检索机器上的 User 资源（资源的首字母须大写）。具体命令及过程如下：

```
# curl -X GET 'http://puppetdb.example.com:8080/v3/resources/User'
# 以下为符合条件的返回结果
[{"parameters" : {
   "uid" : "1000,
   "shell" : "/bin/bash",
   "managehome" : false,
   "gid" : "1000,
   "home" : "/home/foo,
   "groups" : "users,
```

```
    "ensure" : "present"
  },
  "line" : 10,
  "file" : "/etc/puppet/manifests/site.pp",
  "exported" : false,
  "tags" : [ "foo", "bar" ],
  "title" : "foo",
  "type" : "User",
  "certname" : "host1.mydomain.com"
}, {"parameters" : {
    "uid" : "1001,
    "shell" : "/bin/bash",
    "managehome" : false,
    "gid" : "1001,
    "home" : "/home/bar,
    "groups" : "users,
    "ensure" : "present"
  },
  "line" : 20,
  "file" : "/etc/puppet/manifests/site.pp",
  "exported" : false,
  "tags" : [ "foo", "bar" ],
  "title" : "bar",
  "type" : "User",
  "certname" : "host2.mydomain.com"}]
```

案例 2

检索 User 资源为 foo 的标题。具体命令及过程如下：

```
curl -X GET 'http://puppetdb:8080/v3/resources/User/foo'
# 以下为符合条件的返回结果
[{"parameters" : {
    "uid" : "1000,
    "shell" : "/bin/bash",
    "managehome" : false,
    "gid" : "1000,
    "home" : "/home/foo,
    "groups" : "users,
    "ensure" : "present"
  },
  "line" : 10,
  "file" : "/etc/puppet/manifests/site.pp",
  "exported" : false,
  "tags" : [ "foo", "bar" ],
  "title" : "foo",
  "type" : "User",
  "certname" : "host1.mydomain.com"
}]
```

3. Node（节点）类

Node（节点）类可以检索 Node 下的 fact 信息与资源信息，并可以根据表达式对其进行匹配与过滤。首先来看一下 Node 类的路由信息，然后再来看 Node 类的返回信息内容，最后看 Node 类的案例。

以下为 Node 类的路由：

```
# 检索所有 nodes 节点的信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes'
# 检索指定 nodes 节点的信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/<NODE>'
# 检索指定 nodes 节点中的 facts 信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/<NODE>/facts'
# 检索指定 nodes 节点中 facts 的 key 信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/<NODE>/facts/<NAME>'
# 检索指定 nodes 节点中 faces 的 key 对应的 values 信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/<NODE>/
facts/<NAME>/<VALUE>'
# 检索指定 nodes 节点中的 resources 资源类信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/<NODE>/resources'
# 检索指定 nodes 节点下 resources 资源类中的指定资源信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/<NODE>/resources/<TYPE>'
# 检索指定 nodes 节点下 resources 资源类中指定资源对应的标题信息
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/<NODE>/
resources/<TYPE>/<TITLE>'
```

案例 1

检索 web.example.com 的上报信息。具体命令及过程如下：

```
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/web.example.com'
# 以下为符合条件的返回结果
{
  "name" : "web.example.com",
  "deactivated" : null,
  "catalog_timestamp" : "2014-06-14T15:18:51.833Z",      # catalog 环境的环境
  "facts_timestamp" : "2014-06-14T15:38:57.404Z",        # facts 最后的编译时间戳
  "report_timestamp" : null                              # 报告上报时间
}
```

案例 2

检索 web.example.com 的 SElinux 状态。具体命令及过程如下：

```
# curl -X GET 'http://puppetdb.example.com:8080/v3/nodes/web.example.com/facts/
selinux'
# 以下为符合条件的返回结果
[ {
  "certname" : "web.example.com",
```

```
  "name" : "selinux",
  "value" : "false"     # web.exmaple.com 的 selinux 状态为关闭
} ]
```

案例 3

更值得一提的是，可以通过表达式对 node 节点数据进行检索。以下为检索的条件语句。

```
["and",
  ["=", ["fact", "kernel"], "Linux"],
  [">", ["fact", "uptime_days"], 30]]
```

以上条件语句的含义：检索是 Linux 系统的、同时开机大于 30 天的机器的信息。笔者通过一个简单的案例来检索所有上报的 Agent 开机时间大于 10 小时的机器，具体命令如下：

```
# curl -G 'http://puppetdb.example.com:8080/v3/nodes' --data-urlencode '
query=[">",["fact","uptime_hours"],10]'
# 以下为符合条件的返回结果
[ {
  "name" : "sh_adlog_access_storrage.web.example.com"
  "deactivated" : null,
  "catalog_timestamp" : "2014-06-12T15:05:25.349Z",
  "facts_timestamp" : "2014-06-12T15:05:25.064Z",
  "report_timestamp" : "2014-06-12T15:05:23.016Z"
}, {
  "name" : "bj_adlog_access_storrage.web.example.com ",
  "deactivated" : null,
  "catalog_timestamp" : "2014-06-14T15:18:51.833Z",
  "facts_timestamp" : "2014-06-14T15:38:57.404Z",
  "report_timestamp" : null
} ]
```

除了上面我们看到的表达式外，PuppetDB API 还提供了很多其他表达式，如表 14-3 所示。

表 14-3 PuppetDB API 常用表达式

表 达 式	含 义	表 达 式	含 义	表 达 式	含 义
=	等于	>=	大于等于	and	与
>	大于	<=	小于等于	or	或
<	小于	~	正则匹配	not	非

4. Catalogs 类

Catalogs 类可以检索 Hostname 编译后的 LOG。Catalogs 类使用比较简单，其路由格式如下：

```
# curl -X GET 'http://puppetdb.example.com:8080/v3/catalogs/<NODE>' # 其中 node 为
节点名称
```

案例 Catalogs 检索 foo.localdomain 节点的编译日志。具体命令如下：

```
curl -X GET http://puppetdb:8080/v3/catalogs/foo.localdomain
# 以下为符合条件的返回结果
{
  "name" : "yo.delivery.puppetlabs.net",
  "version" : "e4c339f",
  "transaction-uuid" : "53b72442-3b73-11e3-94a8-1b34ef7fdc95",
  "environment" : "production",
  "edges" : [...],
  "resources" : [...],
}
```

5. Reports（报告）类

Reports（报告）类可以检索 Agent 上报到 Master 的报告内容。Reports（报告）类的路由格式如下：

```
# curl -X GET 'http://puppetdb.example.com:8080/v3/reports' # 检索 PuppetDB 中已经上报的报告信息
```

Reports 类检索参数有两个，具体如下。

- certname: 节点名称。
- hash：报告 ID。

案例 检索 example.local 节点的报告，具体命令及过程如下：

```
# curl -G 'http://puppetdb.example.com:8080/v3/reports' --data-urlencode 'query=["=", "certname", "example.local"]'
# 以下为符合条件的返回结果
 [
   # 返回一
 {
   "end-time": "2012-10-29T18:38:01.000Z",     # 上报最后时间
   "puppet-version": "3.0.1",     # Puppet 的版本号
   "receive-time": "2012-10-29T18:38:04.238Z",     # 收到报告的时间
   "configuration-version": "1351535883",     # 配置版本
   "start-time": "2012-10-29T18:38:00.000Z",     # 开始上报时间
   "hash": "bd899b1ee825ec1d2c671fe5541b5c8f4a783472",     # 报告 Hash 值
   "certname": "example.local",     # 报告节点名
   "report-format": 4,     # 报告格式版本
   "transaction-uuid": "030c1717-f175-4644-b048-ac9ea328f221"     # 报告连续性 ID
   },
# 返回二
 {
   "end-time": "2012-10-26T22:39:32.000Z",     # 上报最后时间
   "puppet-version": "3.0.1",     # Puppet 的版本号
   "receive-time": "2012-10-26T22:39:35.305Z",     # 收到报告的时间
   "configuration-version": "1351291174",     # 配置版本
   "start-time": "2012-10-26T22:39:31.000Z",     # 开始上报时间
   "hash": "cd4e5fd8846bac26d15d151664a40e0f2fa600b0",     # 报告 Hash 值
```

```
    "certname": "example.local",     # 报告节点名
    "report-format": 4,     # 报告格式版本
    "transaction-uuid": null     # 报告连续性 ID
    }
]
```

14.4 PuppetDB 问答

在对 PuppetDB 有了深入了解后，很多网友不禁会对 PuppetDB 数据迁移方式、treuststore 和 keystore 文件、PuppetDB 的开发语言和 Puppet apply 编译方式是否支持 PuppetDB 产生一些疑问。以下为 PuppetLabs 官方网站整理的用户对 PuppetDB 的一些疑问与解答。

问题 1：是否可以将数据从 ActiveRecord 的 storeconfigs 或者现有 PuppetDB 迁移到新的实例？

答：可以。在现阶段，只能从 ActiveRecord 迁移输出资源，或者从现有的 PuppetDB 迁移目录。

问题 2：如果 PuppetDB 对 truststore 或 keystore 文件有异议该怎么办？

答：一些原因会导致这一情况的出现，但归根结底是因为 PuppetDB 暂时无法读取用户的信任库。前者的文件包含用户的证书颁发机构的证书，PuppetDB 就用它来验证用户的客户端。后者包含 PuppetDB 用来向用户确认自己的密钥和证书。

简而言之，通常可以通过运行 / usr /sbin/ puppetdb ssl-setup –f 命令来重新初始化自己的 SSL 环境来解决这些问题。但注意这一解决方法的前提是系统已经为 puppet agent 生成了一个整数（也就是：agent 已经运行了一次并且证书已经完成签名）。常见的问题是在 puppet agent 运行前就已经安装了 PuppetDB，上述办法就能解决这一问题以及其他的问题。

详细的回答：如果 puppetdb ssl -setup 命令不能解决你的问题或者你想知道后台发生了什么，你也可以手工管理此配置。你可以在 config 文件中的信任库和密钥库选项中设定信任库和密钥库文件的位置。还应该设置密钥密码和信任密码，确保 keystore.jks 和 truststore.jks 文件保存在 config 文件确认的位置，而且确保它们可以被 PuppetDB 所读取（puppetdb 用于开放源码安装，pe-puppetdb 用于企业版安装）。除此之外，你可以通过使用 keytool -keystore /path/to/keystore.jks 来验证密码是否正确，而且进入密钥密码。同样，可以使用 keytool -keystore /path/to/truststore.jks 来验证信任库。

问题 3：PuppetDB 的 Dashboard 在访问时出现了奇怪的错误。怎么办？

答：通常 Dashboard 会出现两种常见的错误，一种是在非 SSL 下连接；一种是在 SSL 下连接。以下为常见的两种错误的解决方法。

1）非 SSL 下连接：以明文方式连接 8080 端口，但 PuppetDB 没有在监听。

默认情况下，出于安全考虑 PuppetDB 只将明文访问方式监听在本机 localhost（默认 IP:127.0.0.1）域。所以如果我们以明文方式访问 PuppetDB，需要更改 PuppetDB 守护进程

的监听接口，如可以将守护进程监听在可以访问外界网络的网络接口。需要注意的是，因为我们以明文的方式访问，所以需要一种其他的保护手段，如防火墙（IPTABLES）来对网络连接进行限制。

2）通过 SSL 访问 PuppetDB，但是双方没有建立信任关系。

因为 PuppetDB 使用的是 Puppet 基础设施的证书颁发机构，并通过该机构签发的证书，PuppetDB 不信任你的浏览器，你的浏览器不信任 PuppetDB。在这种情况下，就需要给浏览器安装 Puppet 的 CA 签发的证书。

问题 4：PuppetDB 支持 Puppet apply 编译方式吗？

答：支持。关于 puppet apply 方式的更多的信息可以参考 http://docs. puppetlabs.com/puppetdb/latest/connect_puppet_apply.html。

问题 5：为什么 PuppetDB 是用 Java 编写的？

答：其实，PuppetDB 并不是通过 Java 语言开发的，它是以一种叫做 Clojure 的语言编写的，是 Lisp 在 Java 虚拟机上运行的一种方言。其他几种语言是其原型，包括 Ruby 和 JRuby，但这些语言缺乏必要的性能。我们选择 JVM 语言是因为其出色的库功能和高性能。在 JVM 语言中，我们使用 Clojure 是考虑到了它的表现力、性能以及我们团队以往的语言经验。

问题 6：Java 支持哪些版本？

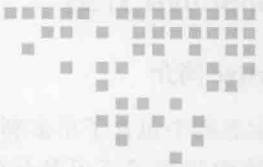
答：正式支持的版本是 OpenJDK 1.7 和 Oracle 的 JDK 1.7。

问题 7：PuppetDB 支持哪些数据库？

答：如果用于生产环境，则推荐使用 PostgreSQL 数据库。另外 PuppetDB 还附带了一个嵌入式的 HyperSQL 数据库，它是一个体积比较小的数据库（我们尚无计划支持其他的数据库，包括 MySQL，虽然 MySQL 使用比较广泛，但它缺乏重要的功能，如数组的列和递归查询）。

问题 8：为什么在装载数据时，数据库服务器上的负载会这么高？

答：数据库服务器上的高负载可能是由很多原因导致的，Puppet 管理的节点的总数量、agent 运行的频率和每次运行节点进行改变的总次数等都有可能导致机器高负载。负载高的一个可能的原因是低 catalog 复制率。你可以查看 PuppetDB 的 Dashboard 来为你的 PuppetDB 实例找到这一复制率。通常该比例应为 90% 或以上。如果这一比例低于 90%，可能会导致数据库更重的 I / O 负载。请参阅故障排除低 catalog 复制率向导的步骤来诊断这一问题。



第 15 章 Chapter 15

Marionette Collective 框架应用

在我们之前所掌握的知识体系结构中，Puppet 是以 C/S 结构主动到 Master 拉取配置信息的。笔者以自己的工作需求为例，Puppet 已经满足了 80% 的应用场景，但是还有 20% 的应用场景是覆盖不到的，如上线一个配置文件需要及时在线生效，而 Puppet 并不会实时拉取配置信息，特别是在 Puppet Agent 比较多的时候，生效的时间更是无法预测，使得这一上线需求并不能很好地完成。其实这种类似的需求可以通过 Marionette Collective（下称 MCollective）来很好地解决，它可以主动推送信息给所有的 Agent，功能类似于 puppet kick 命令，而主动推送信息是 puppet kick 的全部功能，但对于 MCollective 来说这只是其强大功能的冰山一角。MCollective 是一个框架，它的优势是很多功能可以通过插件来实现，也就满足了大多数人在不同场景下的需求。本章就来介绍这个强大的框架，首先介绍 MCollective，让读者对它有一个更深的理解与认识；然后介绍 Middleware（中文翻译“中间件”，下称“中间件”），它是一套降低 MCollective 复杂度的中间层，MCollective 借助它来建立 MCollective Client 与 MCollective Server 的通信；然后介绍 MCollective 环境的安装与配置，由于网路环境的不同，安装的复杂度也是不一样的，读者可以根据自己的情况来选择安装方式，安装与配置的选择主要参照的是网络传输信息的安全级别；接着介绍如何使用 MCollective，并以案例的方式让读者更快地了解它；然后介绍 MCollective 插件应用，这是 MCollective 的核心，很多个性的需求可以通过这一节来解决；最后介绍如何通过 MCollective 管理 Puppet Agent，并通过与 puppet kick 命令的功能进行对比，了解 MCollective 的强大之处。

15.1 MCollective 介绍

1. MCollective 简介

Puppet 的生态圈中包含了很多利器，如 Puppet DB、Facter 和 Puppet Dashboard 等，它们在不同的领域中起到了不可替代的作用。与以上工具相类似，MCollective 也是一个与 Puppet 密切相关的任务编排执行框架，虽然 Puppet 善于管理系统的状态，但是 Agent 默认 30 分钟的运行间隔，使它不适合实时任务的执行与控制，尽管 puppet kick 命令的功能可以在 Master 上主动触发 Agent 的执行，但是它并不适合海量服务器管理的场景。与 puppet kick 命令相比，MCollective 涵盖了 puppet kick 命令的全部功能，同时致力于以一种新颖独特的方式来满足海量服务器管理的需求，它将服务器上报的信息划分在不同的集群中进行管理，在这一点上，它的功能与 Func（https://fedorahosted.org/func/）、Fabric（http://fabfile.org/）和 Capistrano（http://www.capify.org/）相类似。

2. MCollective 的特点

与同类软件 Func、Fabric 和 Capistrano 相比，MCollective 有如下特点：

- 能够与小型、中型到大型服务器集群交互。
- 使用广播范式（broadcast paradigm）来进行请求分发，所有服务器会同时收到请求，而只有与请求所附带的过滤器匹配的服务器才会去执行这些请求。
- 打破了以往用主机名作为身份验证手段的复杂命名规则。使用每台机器自身提供的丰富的目标数据来进行定位。目标数据可以来自于：Puppet、Chef、Facter、Ohai 或者 MCollective 自身提供的插件。
- 使用命令行调用远程代理。
- 能够写自定义的设备报告。
- 大量的代理来管理包、服务和其他来自于社区的通用组件。
- 允许写 SimpleRPC 风格的代理、客户端，以及使用 Ruby 实现 Web UIs。
- 外部可插件化以实现本地需求。
- 中间件系统有着丰富的身份验证、授权模型与数据加密方式，通过这些方式可以建立安全的第一道防线。
- 重用中间件来做集群、路由和网络隔离以实现安全和可扩展安装。

3. MCollective 的优势

使用 MCollective Agent 可以很容易地以其自身数据而不是主机名来划分不同的集群，这意味着我们不需要维护一个庞大的主机名或者 IP 列表，集群中所有的 Agent 都需要实时地汇报自身的信息，有了这些信息后，MCollective 就能将全部的 Agent 划分到不同的集群中，程序也是在一个集群上执行而不是一台机器上执行。

另外MCollective的优势也颠覆了类似通过循环像SSH遍历服务器的方式，因为MCollective使用的RPC框架使我们无须花费太多精力在编写代码连接服务器、将命令传递给各机器以及处理各日志和异常等琐碎的工作上，MCollective可以很好地帮我们解决这些问题。如果想在所有的服务器上执行某动作，就像puppet kick命令那样，可以通过与Puppet结合部署MCollective Server，通过Middleware（中间件）与这些MCollective Server交互传输命令，并执行命令。除此之外MCollective还有以下优势：

- 去中心化存储（No Centralized inventory）。
- 支持成千上万节点（Thousands Of Nodes）。
- 新建框架（Framework For Creation）。
- 异步执行（Asynchronous）。

4. MCollective解决的问题

有许多问题和用例特别适合通过MCollective来解决，如：

- 有多少系统拥有32G的内存。
- 现在线上运行的系统发行版本有哪些。
- 在所有的系统上部署一个1.2.3版本的程序。
- 在质量保障系统上部署一个1.2.4版本的程序。
- 在开发系统上部署一个1.2.5rc2版本的程序。
- 在所有系统上运行Puppet，并确保最多只有10个Puppet同时运行。
- 重启所有位于某IDC的Apache服务器。

5. MCollective工作流程

MCollective通常与Puppet结合使用，MCollective的工作流程如图15-1所示。

从图15-1的MCollective的工作流程中可以看到，MCollective为C/S架构，管理员在MCollective Client上通过MCollective提供的命令通道下发指令给中间件，而MCollective Server通过订阅中间件的消息获取这些指令并执行。整个MCollective的工作流程中共有3个角色，它们分别如下。

- MCollective Client：MCollective的客户端（下称Client），用来发送指令到中间件，MCollective Server通过订阅中间件消息获取指令并在本机执行。通常Client被安装在Puppet Master上，用于发送指令给MCollective Server。
- 中间件：中间件是一种独立的系统软件或服务程序，MCollective借助中间件来搭建Client与MCollective Server连接的桥梁。目前常见的中间件包括ActiveMQ和RabbitMQ。
- MCollective Server：MCollective的服务端（或者被称为node节点，下称Server），通常Server被安装在Puppet Agent上。Server端需要运行MCollectived守护进程来

接收 Client 通过中间件发送的指令，并在 Puppet Agent 上应用这些指令。

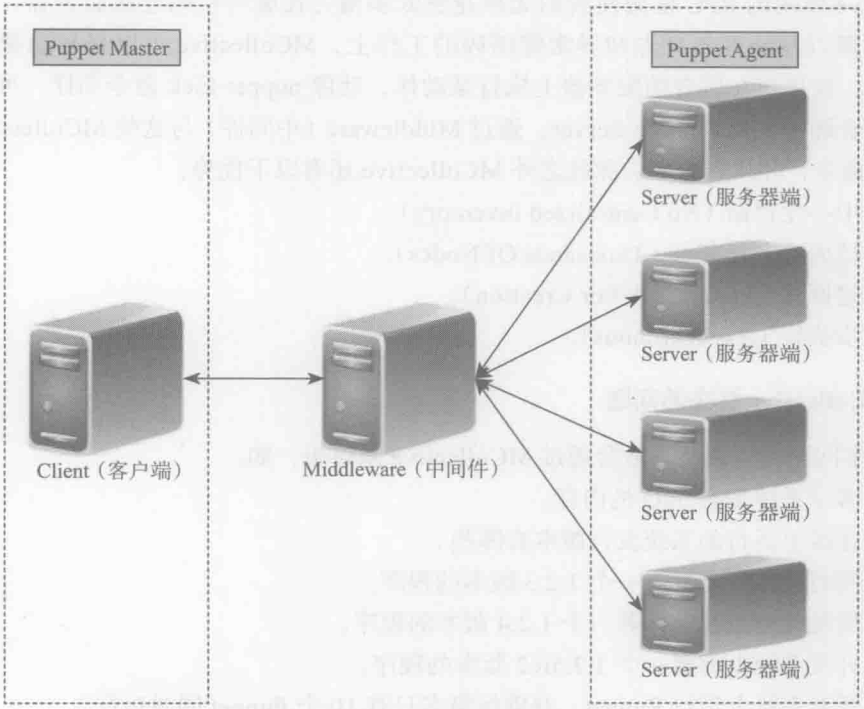


图 15-1 MCollective 工作流程图

目前 MCollective 只提供给我们 API，而尚未提供可视化用户界面，原因是许多强大的命令和控制工具都提供了与 UNIX Shell 相同的功能。虽然 Shell 界面非常强大，但与 API 相比其并不是一个理想的编程接口，而通过 MCollective 的 API 就可以很方便地将编排好的动作通过 Agent 插件在 Agent 上实现，扩展性好，而且简单、方便、实用。这就是 MCollective。

15.2 中间件介绍

上面在介绍 MCollective 的工作流程中，笔者曾提到了中间件。中间件是指在网络环境下，位于操作系统等系统软件和应用软件之间的一种连接分布计算实体的分布式软件，它通过提供简单、一致和集成的分布编程环境，简化分布应用的设计、编程和管理。从本质上讲，中间件是一个分布软件层，屏蔽了底层分布环境（网络、主机、操作系统和编程语言）的复杂性和异构性，主要解决异构网络环境下分布应用软件之间的互联、互通和互操作问题，可以提高应用系统的可移植性。而 MCollective 正是利用了中间件的发布和订阅消息传递技术，这些发布和订阅技术通常使用异步消息软件，如 ActiveMQ 和 RabbitMQ 等来

实现。其实 MCollective 本身使用 Stomp（Streaming Text Orientated Message Protocol，中文译为“流文本定向协议”）来发送和接收消息，所以理论上讲，任何一个实现健壮 Stomp 监听服务的消息中间件都可以与 MCollective 协同工作，只不过 ActiveMQ 和 RabbitMQ 是 MCollective 使用最广泛的两个消息中间件。

15.2.1 ActiveMQ 介绍

ActiveMQ 由 Apache 官方维护，它是一款流行且功能强劲的开源消息总线，运行在 JVM 环境上。ActiveMQ 完全支持 JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现，尽管 JMS 规范出台已经很久了，但是 JMS 在当今的 J2EE 应用中仍然扮演着特殊的地位。ActiveMQ 的特性如下：

- 可以使用多种语言和协议编写客户端。语言：Java、C、C++、Ruby、Perl、Python 和 PHP 等。应用协议：OpenWire、Stomp、REST、WS Notification、XMPP 和 AMQP 等。
- 完全支持 JMS1.1 和 J2EE 1.4 规范。
- 支持 Spring，ActiveMQ 可以很容易内嵌到使用 Spring 的系统里面去，而且也支持 Spring 2.0 的特性。
- 通过了常见 J2EE 服务器（如 Geronimo、JBoss 4、GlassFish 和 WebLogic）的测试，其中通过 JCA 1.5 resource adaptors 的配置，可以让 ActiveMQ 自动部署到任何兼容 J2EE 1.4 的商业服务器上。
- 支持多种传送协议，如 in-VM、TCP、SSL、NIO、UDP、JGroups 和 JXTA 等。
- 支持通过 JDBC 和 journal 方式提供高速的消息持久化。
- 从设计上保证了高性能的集群、客户端到服务器或者点对点。
- 支持 Ajax。
- 支持与 Axis（全称为 Apache EXtensible Interaction System）的整合。
- 可以很容易地调用内嵌 JMS provider 进行测试。

关于 ActiveMQ 的更多信息请参考官方网站 http://activemq.apache.org/。

15.2.2 RabbitMQ 介绍

RabbitMQ 是一个实现了高级消息排队协议（AMQP）的消息队列服务。RabbitMQ 基于 OTP（Open Telecom Platform，开放电信平台）进行构建，并使用 Erlang 语言和运行环境来实现。关于 RabbitMQ 的更多信息请参考官方网站 http://www.rabbitmq.com/。

15.3 MCollective 环境的安装与配置

MCollective 由 Puppet 官方提供开发与维护，2009 年 12 月 2 日发布了 MCollective 首

个旗舰版本，截止到本书出版前历经了30多个大小版本。我们可以从版本变更记录http://docs.puppetlabs.com/MCollective/releasenotes.html上的内容对其进行了解。MCollective的功能从单一到逐渐丰富并不断保持它的活力。目前MCollective可以运行在UNIX/Linux系统上，也可以运行在微软的Windows系列操作系统上，由于它支持的操作系统发行版本比较多，不逐一介绍，可以通过http://docs.puppetlabs.com/MCollective/deploy/install.html#system-requirements进行了解。与Puppet一样，MCollective也是通过Ruby语言编写的，因为Ruby本身就可以跨越平台，所以它也继承了这个优势，不局限对某一系统的应用。但需要注意的是，MCollective与Puppet也有着相同的弊病，那就是它并不支持Ruby所有的版本。如目前MCollective还没有正式提供对Ruby2版本的支持，另外在MCollective连接中间件时，因为Ruby1.8.5和Ruby1.8.6版本不支持TLS（安全传输层协议），所以也不能使用。Ruby1.9.0到1.9.2版本对MCollective的支持也不是很好，唯有Ruby 1.9.3和Ruby1.8.7版本适用于MCollective。这里推荐使用Ruby1.8.7版本，因为它对Puppet也做了很好的支持。另外还需要注意，因为MCollective通过Stomp协议进行通信，所以在安装Ruby环境后还要安装Stomp。笔者在整个配置环境中使用了CentOS 6.5_84的操作系统环境。下面首先从MCollective的安装开始介绍，让读者了解MCollective的角色与分工；再介绍中间件的安装与配置，读者需要注意线上运行的MCollective只需要一个消息中间件，可以选择ActiveMQ或者RabbitMQ，中间件的配置复杂程度不一样，需要根据自己的网络安全状况级别来选择适合的配置方式。

15.3.1 MCollective安装

MCollective安装时需要注意，MCollective角色不同安装的软件也不一致，严格的安装需要注意以下事项：

- ❑ Server端必须安装mcollective软件包，通常Server与Puppet Agent运行在同一台服务器上。
- ❑ Client端必须安装mcollective-client软件包，通常Client与Puppet Master运行在同一台服务器上。
- ❑ Server端与Client端都需要安装mcollective-common软件包。

但笔者更推荐大家不区分Server或Client，这些包都安装，安装包彼此间也不会有什么冲突。下面来介绍MCollective的两种安装方式。

方式1 在常见的系统发行版本（如Debian、Ubuntu、RedHat和CentOS系统发行版本）上安装MCollective。如果读者的操作系统环境不在上述常见系统发行版本中，可以参考http://docs.puppetlabs.com/MCollective/deploy/install.html#running-from-source，通过源码方式来安装MCollective。

1）在Debian和Ubuntu上安装MCollective环境。根据上面介绍的3个角色，我们分别在Client与Server上进行安装，下载MCollective的软件包并安装，具体命令如下：

```
    # 下载
    # wget http://downloads.puppetlabs.com/MCollective/MCollective_2.0.0-1_all.deb
    # wget http://downloads.puppetlabs.com/MCollective/MCollective-client_2.0.0-2_
all.deb
    # wget http://downloads.puppetlabs.com/MCollective/MCollective-client_2.0.0-2_
all.deb
    安装
    # dpkg -I MCollective*.deb
```

MCollective 安装完成后，接着安装 Stomp 软件包，因为 MCollective 需要通过 Stomp 协议进行通信（Stomp 需要安装 1.2.2 或者更高版本）。

```
    # apt-get ruby-stomp
```

2）在 RedHat 和 CentOS 上安装 MCollective，同样分别在 Client 与 Server 上进行安装。首先安装 Puppet 官网提供的软件源。

```
    # 安装 Puppet 官方源
    # rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
    # 导入 GPG 密钥
    # rpm -import http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
```

3）安装 MCollective 的软件包（包含 mcollective、MCollective-client 和 mcollective-common 软件包），通过通配符“*”安装 MCollective 的相关环境。

```
    # yum install MCollective-*
```

最后安装 rubygem-stomp（Stomp 需要安装 1.2.2 或者更高版本）。

```
    # yum install rubygem-stomp-1.3.2
```

方式 2 通过 Puppet 来部署 MCollective（推荐使用）。当我们管理的服务器比较多的时候，通过 Puppet 来部署 Stomp 与 MCollective 是一个不错的选择，以下为部署 MCollective 的基础模块的代码，它包含了 Stomp 和 MCollective 环境的安装，以及 MCollective 的守护进程的启动（MCollective 守护进程需要启动在 Server 端上）。

```
    class MCollective {
      # Stomp 软件包安装（Stomp 需要安装 1.2.2 或者更高版本）
      package {'stomp':
       ensure   => '1.2.2',
       provider => gem,
       before   => Package['MCollective'],
      }
      # mcollective 环境安装
      package {'MCollective':
       ensure => latest,
      }
      # mcollective 启动
      service {'MCollective':
```

```
    ensure  => running,
    enable  => true,
    require => Package['MCollective'],
  }
}
```

15.3.2 MCollective 配置

本节主要介绍 Client、中间件与 Server 的配置过程。由于配置的环节比较多，为了让读者更清晰地掌握 MCollective 配置过程，将整个 MCollective 配置信息以表列出，如表 15-1 所示。

表 15-1 MCollective 与中间件配置信息

角 色	Hostname（主机名）	IP
Client 与 Puppet Master	master.example.com	192.168.7.7
中间件（ActiveMQ/RabbitMQ）	mq.example.com	192.168.7.8
Server 与 Puppet Agent	agent.exmaple.com	192.168.7.9

关于中间件的使用，MCollective 官方推荐我们使用 ActiveMQ。ActiveMQ 优势是：很多网络测试表明，ActiveMQ 有着良好的性能，并且足够安全与灵活。但它也有缺点，它使用笨拙的 XML 配置文件，在复杂的部署中可能需要编辑多个区段。但这并不影响我们使用，所以本节主要通过 ActiveMQ 搭建 Client 与 Server 的桥梁。关于 RabbitMQ 的配置，如果读者有需求可以参考官方网站 http://docs.puppetlabs.com/MCollective/reference/plugins/connector_rabbitmq.html 获取更多配置信息，笔者在此就不深入介绍 RabbitMQ 了。

另外由于我们使用了最新版本的 MCollective/ActiveMQ 接口，如果服务器上已经安装了 MCollective/ActiveMQ 环境，在使用前需要注意它们的版本情况。

- MCollective 需要升级到 2.0.0.0 或更高版本。
- ActiveMQ 需要升级到 5.5.0 或更高版本。
- Stomp gem 需要升级到 1.2.2 或更高版本。

现在开始部署 MCollective 与 ActiveMQ，目前有两种配置方案，读者可以根据自己的网络状况来选择不同的方案。

方案 1 MCollective 通过 ActiveMQ 提供的账号和密码进行连接，此方式的优势是配置简单，劣势是明文传输安全性较差，适合在比较安全的网络环境中使用。

笔者通过以下 5 步来配置非加密连接的方案 1。

1）在主机 mq.example.com（IP：192.168.7.8）上安装 ActiveMQ 软件包。由于默认系统源并不包括 ActiveMQ 软件包，需要借助 Puppet 源来安装 ActiveMQ。

```
# 软件源的安装
# rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
# 安装 ActiveMQ
```

```
# yum install activemq
```

2）配置 ActiveMQ，在 ActiveMQ 中创建 MCollective 的 Server/Client 连接时用的账号与密码。编辑 ActiveMQ 的 /etc/activemq/activemq.xml 配置文件，查找 simpleAuthenticationPlugin 关键字，修改文件中的默认账号与密码（修改前建议备份 activemq.xml 文件，以防改错启动失败）。关于 ActiveMQ 的配置大家还可以参考网上的文章 http://manzuosteve.blog.51cto.com/1966239/1192190。

```
<plugins>
     <simpleAuthenticationPlugin>
       <users>
        <authenticationUser username="admin" password="secret" groups="MCollective,
admins,everyone"/>     # 将 username 由 admin 替换为 mcollective，密码由 secret 替换为 mypass
       </users>
     </simpleAuthenticationPlugin>
 </plugins>
```

确认 ActiveMQ 的 61613 端口在 activemq.xml 配置文件中处于开启状态。

```
<transportConnectors>
      <transportConnector name="stomp+nio" uri="stomp+nio://0.0.0.0:61613"/>
</transportConnectors>
```

然后启动 ActiveMQ 的守护进程。

```
# service activemq start
```

最后通过 netstat –tnl 命令确认 61613 端口处于监听状态。如果端口处于监听状态说明启动守护进程成功。

```
# netstat -tnl | grep 61613
```

3）配置 Client 端（主机 master.example.com，IP：192.168.7.7），Client 的功能是发送指令到 ActiveMQ。修改 Client 端 /etc/MCollective/client.cfg 配置文件，使 Client 端与 ActiveMQ 建立联系。为了介绍方便，笔者将 Client.cfg 配置文件分为了 5 个部分（具体见代码中的注释），读者可以重点关注第四部分（Client.cfg 的大部分参数可以保持默认），Client.cfg 常用配置如下。关于 Client.cfg 配置文件的更多信息请参考官方网站 http://docs.puppetlabs.com/MCollective/configure/client.html。

```
# client.cfg
# 1)Subcollectives 部分
main_collective = MCollective
collectives = MCollective
# 2) 基本配置部分
libdir = /usr/libexec/MCollective
logger_type = file
logfile = /var/log/MCollective.log
```

```
loglevel = warn
# 3) 安全部分
securityprovider = psk
plugin.psk = unset
# 4) MQ 消息队列连接设置部分
connector = activemq    # 连接插件，可以使之 ActiveMQ 或 RabbitMQ
plugin.activemq.pool.size = 1       # ActiveMQ 的 ID
plugin.activemq.pool.1.host = mq.example.com    # ActiveMQ 的地址
plugin.activemq.pool.1.port = 61613     # 非加密的 Stomp 协议
plugin.activemq.pool.1.user = MCollective     # 连接 ActiveMQ 的用户名
plugin.activemq.pool.1.password = mypass      # 连接 ActiveMQ 的密码
# 5) Facts 部分
factsource = yaml
plugin.yaml = /etc/MCollective/facts.yaml
fact_cache_time = 300
```

如以上配置所示，配置文件共分为以下 5 个部分。

- Subcollectives 部分。
  - main_collective 参数：默认请求策略。
  - collectives 参数：分类请求策略，默认情况下，所有的 Server 属于单一广播域，可以追加 bj_MCollective、sz_ MCollective、sh_ mcollective 和 gz_ mcollective 等，以地域划分接收请求，关于广播域的划分可以参考官方网站 http://docs.puppetlabs.com/MCollective/reference/basic/subcollectives.html。
- 基本设置部分。
  - libdir 参数：插件搜索路径。MCollective 在不同操作系统发行版本中插件的位置也不一样，如表 15-2 所示。当然也可以通过 libdir 参数改变插件的存放位置。

表 15-2 MCollective 插件默认存放目录

操作系统发行版本	插件默认目录
Red Hat 系列操作系统	/usr/libexec/MCollective
Debian 系列操作系统	/usr/share/MCollective/plugins

  - logger_type 参数：MCollective 守护进程日志记录方式，默认为 file（文件记录），可选参数 syslog 与 console。
  - logfile 参数：指定 log 写入文件地址。
  - loglevel 参数：记录日志级别，默认为 info。可选 fatal、error、warn、info 和 debug。
- 安全部分。
  - securityprovider 参数：使用哪种安全插件，默认值为 psk，它是一个共享的密码，用于服务器间交换数据凭证。securityprovider 参数还有两个可选值 ssl 和 aes_security。
  - plugin.psk 参数：共享密码。
- MQ 消息队列连接设置部分。
  - connector 参数：连接的中间件，默认为 ActiveMQ，可选参数 RabbitMQ。

- plugin.activemq.pool.size 参数：为 ActiveMQ 的池子 ID 号，值为 1，以下配置参数均对池子 ID1 生效。
- plugin.activemq.pool.1.host 参数：指定中间件的服务器地址。
- plugin.activemq.pool.1.port 参数：根据中间件的配置指定连接的端口。
- plugin.activemq.pool.1.user 参数：指定访问中间件的账户名。
- plugin.activemq.pool.1.password 参数：指定中间件的账户密码。

❑ Facts 部分。

- factsource 参数：指定 Facts 插件，默认为 yaml。
- plugin.yaml 参数：缓存 Facts 的配置文件。
- fact_cache_time 参数：Facts 缓存时间，单位秒。

4）配置 Server 端（主机 agent.exmaple.com，IP：192.168.7.9），它以 MCollective 守护进程的方式运行在 Puppet Agent 端，当 ActiveMQ 收到 Client 的指令，Server 会通过订阅消息的方式从 ActiveMQ 获取指令信息，并在 Puppet Agent 端执行。我们要指定 Server 端配置文件到 ActiveMQ，编辑 Server 端的配置文件 /etc/MCollective/server.cfg，然后启动 MCollective 守护进程。由于 Client.cfg 与刚介绍的 Server.cfg 很多配置参数一致配置，所以为了简化流程，笔者通过注释的方式只介绍两个配置文件的差异部分，读者需要重点关注差异部分与 ActiveMQ 的连接部分。关于 Server.cfg 配置文件的更多信息请参考官方网站 http://docs.puppetlabs.com/MCollective/configure/server.html。

```
# server.cfg
main_collective = MCollective
collectives = MCollective
libdir = /usr/libexec/MCollective
logfile = /var/log/MCollective.log
loglevel = info
daemonize = 1     # 是否在后台运行 mcollective 守护进程
securityprovider = psk
plugin.psk = unset     # 共享密码
connector = activemq     # 连接插件，可以使用 ActiveMQ 或 RabbitMQ
plugin.activemq.pool.size = 1     # ActiveMQ 的 ID
plugin.activemq.pool.1.host = mq.example.com     # ActiveMQ 的地址
plugin.activemq.pool.1.port = 61613     # 非加密的 Stomp 协议
plugin.activemq.pool.1.user = MCollective     # 连接 ActiveMQ 的用户名
plugin.activemq.pool.1.password = mypass     # 连接 ActiveMQ 的密码
factsource = yaml
plugin.yaml = /etc/MCollective/facts.yaml
fact_cache_time = 300
```

读者需要注意 Server 端与 Client 的区别，在 Server 端编辑 Server.cfg 配置文件后，要启动 MCollective 守护进程。每次修改 Server.cfg 配置文件要重新启动 MCollective 守护进程（当服务器较多时，推荐使用 Puppet 对 Server 的 MCollective 守护进程进行配置与管

理）。以下为启动 MCollective 守护进程的方法。

```
# service MCollective start
```

5）MCollective 与 ActiveMQ 环境配置完毕后，在 Client 端（主机 master.example.com，IP：192.168.7.7）通过 MCollective 自带 mco 命令的 ping 子命令进行最终配置结果的测试。

```
# mco ping
master.example.com                    time=85.78 ms
agent.example.com                     time=126.42 ms
---- ping statistics ----
2 replies max: 126.42 min: 85.78 avg: 106.10
```

mco 命令下的 ping 子命令的功能是让 MCollective 框架中所有运行的 Server 端做出反馈，它可以测试 Server 端、ActiveMQ 和 Client 配置是否正常，所以以上输出表明我们已经成功地配置了 MCollective。如果 mco ping 命令没有返回结果，很有可能是由以下两种原因导致的：

- Client 端与 Server 端密钥不匹配。
- Client 端或服务端中的 Stomp 用户名或密码不正确。

如果并不是以上原因导致的，我们还可以从 /var/log/activemq/activemq.log 日志中查找线索，并妥善解决。

方案 2 使用安全传输层协议（TLS），用于在两个通信应用程序之间提供保密性和数据完整性。此方式的优势是安全性较高，适用于安全性较差的网络环境，劣势是配置比较复杂。

笔者通过 9 步来配置安全传输层协议（TLS）的方案 2。整个配置过程中，使用了 Puppet 的证书来配置安全传输层协议（TLS）。

1）在 master.example.com（IP: 192.168.7.7）上找到 Puppet Master 的 CA、密钥和根证书的位置，通常它们会被放置在以下位置。

```
# /var/lib/puppet/ssl/certs/ca.pem
# /var/lib/puppet/ssl/certs/master.example.com.pem_cert.pem
# /var/lib/puppet/ssl/private_keys/master.example.com.pem_private.pem
```

以上 Puppet Master 证书为默认配置目录，如果无法在默认目录找到，我们还可以通过 Puppet 提供的 puppet agent --configprint ssldir 命令找到它们。

2）在主机 mq.example.com（IP：192.168.7.8）上创建 ssl 目录，通过 scp 命令复制 Puppet Master 证书文件到 ssl 目录。

```
# 创建 ssl 目录
# mkdir -p /etc/activemq/ssl/
# 复制 ca.pem 文件
# scp root@192.168.7.7: /var/lib/puppet/ssl/certs/ca.pem  /etc/activemq/ssl/
# 复制 master.example.com.pem_cert.pem 证书文件
```

```
# scp root@192.168.7.7: /var/lib/puppet/ssl/certs/master.example.com.pem_cert.pem
/etc/activemq/ssl/
# 复制 master.example.com.pem_private.pem 证书文件
# scp root@192.168.7.7: /var/lib/puppet/ssl/private_keys/master.example.com.pem_
private.pem /etc/activemq/ssl/
```

复制证书后，需要确认文件证书是否有读取权限。通过以下命令授权目录与目录中的文件。

```
# chmod -R  766 /etc/activemq/ssl/
```

3）在主机 mq.example.com（IP：192.168.7.8）上创建 Truststore 文件。Truststore 文件决定哪些证书可以连接到 ActiveMQ 上，如果导入的是某个 CA 认证，ActiveMQ 将会信任由这个 CA 签名的任何证书。稍后我们还需创建 keystore 文件，两个文件创建时都需要密码，这里我们统一将密码设为“mypass”。

```
# keytool -import -alias "My CA" -file ca.pem -keystore truststore.jks
```

校验 Truststore 文件。

```
# keytool -list -keystore truststore.jks
# openssl x509 -in ca.pem -fingerprint -md5
```

4）创建 Keystore 文件。Keystore 包含 ActiveMQ 的证书和密钥，用来向连接它的应用程序标识它自身。

```
# cat master.example.com_private.pem  master.example.com_cert.pem > temp.pem
# openssl pkcs12 -export -in temp.pem -out activemq.p12 -name master.example.com
# keytool -importkeystore -destkeystore keystore.jks -srckeystore activemq.p12 -
# srcstoretype PKCS12 -alias master.example.com
```

校验 Keystore 文件。

```
# keytool -list -keystore keystore.jks
# openssl x509 -in puppet.example.com_cert.pem -fingerprint -md5
```

5）将生成的 Truststore 文件与 Keystore 文件移动到 /etc/activemq/ 目录。

```
# mv keystore.jks trustore.jks /etc/activemq/
```

将 Truststore 文件与 Keystore 配置文件加入 ActiveMQ 的 /etc/activemq/activemq.xml 配置文件中。需要注意将以下配置追加在 Plugins 和 systemUsage 两个标签中间。

```
<sslContext>
    <sslContext
      keyStore="keystore.jks" keyStorePassword="secret"
      trustStore="truststore.jks" trustStorePassword="secret"
    />
 </sslContext>
```

6）在 ActiveMQ（主机 mq.example.com，IP：192.168.7.8）中配置监听端口。MCollective 与 ActiveMQ 不同协议的连接方式通过端口来区分，如下：

- 61613 端口用于非加密的 Stomp。
- 61614 端口用于加密的 Stomp 和 TLS。
- 61616 端口用于非加密的 OpenWire。
- 61617 端口用于 TLS 和 OpenWire。

编辑 ActiveMQ 配置文件 /etc/activemq/activemq.xml，追加以下内容到 transportConnectors 中。开启加密的 Stomp 和 TLS 协议。

```
    <transportConnectors>
     <transportConnector name="stomp+ssl" uri="stomp+ssl://0.0.0.0:61614?needClient-
Auth=true"/>
    </transportConnectors>
```

启动 ActiveMQ 的守护进程。

```
# service activemq start
```

通过 netstat-tnl 命令确认 61614 端口处于监听状态。如果处于监听状态说明启动成功。

```
# netstat -tnl | grep 61614
```

7）配置 Client 端（主机 master.example.com，IP：192.168.7.7），让 Client 端与 ActiveMQ 建立联系，同时配置 Client 与 Puppet Master 共用相同的证书。编辑 Client 端的配置文件 /etc/MCollective/client.cfg。具体如下：

```
# client.cfg
connector = activemq
securityprovider = ssl    # 加密连接方式
# Optional:
plugin.activemq.base64 = yes
plugin.activemq.pool.size = 1
plugin.activemq.pool.1.host = mq.example.com
plugin.activemq.pool.1.port = 61614      # 加密的 Stomp 和 TLS
plugin.activemq.pool.1.user = MCollective      # 连接 ActiveMQ 的账号
plugin.activemq.pool.1.password = mypass      # 连接 ActiveMQ 的密码
plugin.activemq.pool.1.ssl = true
plugin.activemq.pool.1.ssl.ca = /etc/activemq/ssl/ca_crt.pem
plugin.activemq.pool.1.ssl.key =/var/lib/puppet/ssl/private_keys/master.example.
com.pem
plugin.activemq.pool.1.ssl.cert = /etc/activemq/ssl/master.example.com.pem
```

8）配置 Server 端（主机 agent.exmaple.com，IP：192.168.7.9）。让 Server 端与 ActiveMQ 建立联系，同样配置 Server 与 Puppet Master 共用相同的证书。编辑 Client 端的配置文件 /etc/MCollective/server.cfg。具体命令如下：

```
# server.crg
connector = activemq
securityprovider = ssl
# Optional:
plugin.activemq.base64 = yes
plugin.activemq.pool.size = 1
plugin.activemq.pool.1.host = mq.example.com
plugin.activemq.pool.1.port = 61614      # 加密的 Stomp 和 TLS
plugin.activemq.pool.1.user = MCollective     # 连接 ActiveMQ 的账号
plugin.activemq.pool.1.password = mypass     # 连接 ActiveMQ 的密码
plugin.activemq.pool.1.ssl = true
plugin.activemq.pool.1.ssl.ca = /var/lib/puppet/ssl/ca/ca_crt.pem
plugin.activemq.pool.1.ssl.key = /var/lib/puppet/ssl/private_keys/master.example.
com.pem
plugin.activemq.pool.1.ssl.cert = /var/lib/puppet/ssl/certs/master.example.com.pem
```

启动 Server 端守护进程。

```
# service MCollective start
```

9）MCollective 与 ActiveMQ 配置完毕后，再次通过 mco 命令的 ping 参数对最终配置结果进行测试。具体方法如下：

```
# mco ping
master.example.com                    time=85.78 ms
agent.example.com                     time=126.42 ms
---- ping statistics ----
2 replies max: 126.42 min: 85.78 avg: 106.10
```

到目前为止已经介绍了 MCollective 的两种配置方式，接下来将介绍如何使用 MCollective。

15.4 如何使用 MCollective

MCollective 以 Client-> 中间件 ->Server 的方式运行，我们可以通过 MCollective 提供的集成 mco 命令并借助中间件向所有的 Server 发送指令。本节将介绍 MCollective 命令的使用。首先介绍 MCollective 基础命令，主要让读者对命令的使用与参数结构有个基本的了解；然后介绍 MCollective 插件应用，其实除了建立 Client 与所有 Server 的桥梁以外，MCollective 更强大的地方是插件的使用；最后介绍如何通过 MCollective 管理 Puppet Agent。

15.4.1 MCollective 基础命令

1. MCollective 基础命令

Client 通过 mco 命令与所有的 Server 进行交互。通过 mco help 命令可以查到它的子命

令，如下：

```
# mco help
Marionette Collective 版本 2.5.2
  completion      #shell 辅助系统
  facts            # 使用 facts 信息显示报告
  filemgr          # 文件管理客户端
  find             # 通过过滤条件查找主机
  help             # 显示帮助文档
  inventory       # 来自 nodes、collectives 和 subcollectives 的报告工具
  iptables        # 系统防火墙客户端
  nettest          # 网络节点测试工具
  nrpe             # 客户端到 Nagios 的远程插件执行系统
  package          # 包管理工具，包括安装、卸载和升级
  ping             # ping 所有节点
  plugin           # MCollective 插件管理
  puppet           # 管理 Puppet Agent 程序
  rpc              # 通过 RPC 进行应用程序交互
  service          # 利用 service 管理系统服务，包括启动、关闭、重启和查看状态等
```

mco 命令的子命令参数通过 help 参数可以查看到其使用方法。

```
# mco help rpc
Generic RPC agent client application
Usage: mco rpc [options] [filters] --agent <agent> --action <action> [--argument
<key=val> --argument ...]
Usage: mco rpc [options] [filters] <agent> <action> [<key=val> <key=val> ...]...
```

由于 mco package 子命令输出的参数较多，笔者对以上内容作了截取，具体的命令使用方式将通过后续的案例详细介绍。从上面我们可以看到 mco 命令的使用方式与 Puppet 的命令使用方式一致，这也降低了我们学习的成本。

2. 远程执行 RPC 请求

MCollective 远程命令的执行通过 RPC 实现。RPC(Remote Procedure Call Protocol)——远程过程调用协议，它通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。在 MCollective 上通过 mco rpc 子命令方式支持此 RPC 协议，另外 mco rpc 命令也能够与很多标准远程过程调用代理进行交互。以下为使用案例，在 Client 上通过 RPC 命令向所有的 Server 发送指令，如让 Server 启动 Apache，命令及输出结果如下：

```
# mco rpc service start service=httpd
Discovering hosts using the mc method for 2 second(s) ... 2
 * [ ============================================================> ] 2 / 2
web.example.com  Request Aborted    # 响应主机的 Hostname
Summary of Service Status:
  running = 1
  unknown = 1
Finished processing 2 / 2 hosts in 969.38 ms
```

如以上 mco rpc 命令输出所示，mco rpc 命令已经启动主机 web.example.com 的 Apache。mco 命令的整个执行流程如下。

1）在网络发现并找到多台 Server。

2）Client 发送请求给 ActiveMQ，Server 获得请求指定，并显示回复的进度条。

3）Server 显示出现的结果，如执行成功或失败，如为失败则显示失败原因。

4）在 Client 显示执行的统计信息。

3. 远程安装软件

MCollective 还支持远程服务的安装、更新与卸载。如在所有的 Server 上安装 tree 命令。

```
# mco package install tree
Do you really want to operate on packages unfiltered? (y/n): y
 * [ ============================================================> ] 2 / 2
Summary of Ensure:
 1.5.3-2.el6 = 2
```

查看所有 MCollective Server 上 tree 命令的安装状态。

```
# mco package status tree
 * [ ============================================================> ] 2 / 2
          master:example.com: tree-1.5.3-2.el6.x86_64     # 相应的服务器 Hostname
          web.example.com: tree-1.5.3-2.el6.x86_64        # 相应的服务器 Hostname
Summary of Arch:
   x86_64 = 2
Summary of Ensure:
   1.5.3-2.el6 = 2
```

如以上命令输出所示，我们已经在 master:example.com 与 web.example.com 安装了 tree 命令。关于 mco package 还有很多的子参数，可以通过 mco help package 查找到。

4. 过滤器的使用

MCollective 每个子命令都提供了 Filters(中文译为“过滤器”，下称“过滤器”）的功能，用于将某一类机器作为一个整体来对其进行操作。以下以 ping 子命令为例：

```
# mco help ping
Host Filters
   -W, --with FILTER               Combined classes and facts filter
   -S, --select FILTER             Compound filter combining facts and classes
   -F, --wf, --with-fact fact=val  Match hosts with a certain fact
   -C, --wc, --with-class CLASS    Match hosts with a certain config management
class
   -A, --wa, --with-agent AGENT    Match hosts with a certain agent
   -I, --wi, --with-identity IDENT Match hosts with a certain configured identity
```

可以看到过滤器的参数可以匹配 fact、class、agent 和“/”等。以下为过滤器的常用参

数案例与复合语句案例。

```
# 匹配 facts 的 country 为 de，并在这些机器上安装 zsh 软件包
# mco rpc package install zsh -F country=de
# 匹配 kernel 版本为 2.6 同时 cpu 个数大于或等于 2 个的机器，并主动触发 Puppet Agent 运行一次
# mco puppet -v runonce rpc --np -F kernelmajversion='2.6' -F physicalpro-
cessorcount=>'2'
# ping 所有服务中的 Server
# mco ping -A service
# mco ping --with-agent service
# ping 所有包含 apache 类的机器
# mco ping -C apache
# mco ping --with-class apache
# ping 所有符合正则表达式的机器
# mco ping -C /service/
# 指定在 a.example.com 上安装 vsfptd 软件
# mco package install vsftpd -I /a.example.com$/
```

5. 复杂查询案例

1）查找 facts 为 acme 在 staging 环境下的机器或者为 development 环境机器，同时匹配 apache 类。

```
# mco ping -S "((customer=acme and environment=staging) or environment=development)
and /apache/"
```

2）查找为开发环境，但 facts 的 customer 不包含 acme 的机器。

```
mco ping -S "environment=development and !customer=acme"
```

15.4.2 MCollective 插件应用

MCollective 是一个框架，大部分的功能都是通过插件完成的，而 Puppet 官方提供了很多插件供我们选择，可以在官方网站 http://projects.puppetlabs.com/projects/MCollective-plugins/wiki 查找到关于 MCollective 的一些常用插件，更多的插件可以通过 https://github.com/puppetlabs/MCollective-plugins.git 获取。这里介绍一个比较实用的插件案例，使读者了解插件是如何获取使用的。笔者在 Client 上通过“Shell 命令查询所有 Server”状态的插件，此插件并非由官方提供，可以在 https://github.com/cegeka/MCollective-shell-agent 找到它。

首先克隆 Shell 插件的源文件到本地。

```
# git clone https://github.com/cegeka/MCollective-shell-agent.git
```

然后将克隆后的 Shell 插件复制到 MCollectived 的 Server 和 Client 插件目录。

```
# cd MCollective-shell-agent
# cp agent/shell.ddl shell.rb  /usr/libexec/MCollective/MCollective/agent/
# cp application/shell.rb  /usr/libexec/MCollective/MCollective/application/
```

如果不确定 MCollective 插件目录的位置，可以通过 grep"libdir"/etc/MCollective/client.cfg 进行查找。插件包含了两个 Ruby 文件和一个 DLL 文件，DLL 文件提供了插件接收传入参数的具体描述信息。

在 Server 上安装插件后不要忘记让守护进程重新加载配置。具体加载方法如下：

```
# /etc/rc.d/init.d/mcollective reload-agents
```

最后在 Client 中使用 Shell 插件查看 Server 机器的负载情况。

```
# mco shell uptime
Discovering hosts using the mc method for 2 second(s) ... 1
================================================================================
Host: web.example.com
Exitcode: 0
================================================================================
Output:
 08:02:14 up  1:05,  1 user,  load average: 0.05, 0.01, 0.00
================================================================================
```

如果网上的插件无法满足需求，也可以根据自身的实际情况与应用场景来独立开发插件。关于 MCollective 插件开发的更多的信息请参考官方网站 http://docs.puppetlabs.com/MCollective/simplerpc/agents.html。

15.4.3 通过 MCollective 管理 Puppet Agent

Puppet 3.* 版本已经废弃了对 puppet kick（客户端主动更新工具）的支持，如果读者使用的是 Puppet 3.* 版本可以借助 MCollective 的 Puppet Agent 插件来弥补这一缺失。MCollective 的 Puppet Agent 插件安装后，可以管理所有 Puppet Agent。首先通过 MCollective 主动触发所有的 Puppet Agent 运行一次。

```
# mco puppet runonce
 * [ ============================================================> ] 2 / 2
web.example.com              Request Aborted
   Puppet is currently applying a catalog, cannot run now
email.example.com                Request Aborted
   Puppet is currently applying a catalog, cannot run now
Finished processing 2 / 2 hosts in 218.45 ms
```

如以上输出所示，MCollective 通过 Puppet Agent 插件主动触发所有的 Puppet Agent 执行一次，执行后的机器会输出到终端上。当我们使用 MCollective 运行所有的 Puppet Agent 时，所有的 Puppet Agent 上所有的守护进程都是处于关闭状态的，这时如果又在 Puppet Agent 上运行客户端就会显示 notice: Run of Puppet configuration client already in progress; skipping 的警告信息，而这样同时运行 Agent 只有一次机会让 Agent 的 catalog 生效，机会落到哪个 Puppet Agent 上这取决于谁先触发 Puppet Agent 的执行。所以我们通过 crontab

定时运行 Puppet Agent，又通过 MCollective 的插件再次运行 Puppet Agent 时要注意它的时间差问题。那么 Puppet Agent 根据什么来判断本身守护进程是否在执行呢？Puppet Agent 运行时会在 /var/lib/puppet/state/puppetlock（我们可以通过 puppet agent --configprint puppetdlockfile 命令找到它）下生成一个锁文件，此文件用于标识进程运行中的状态，多个 Puppet Agent 会检查此文件是否存在，如果存在就会出现刚刚报的 notice: Run of Puppet configuration client already in progress; skipping 警告信息。

除此之外 MCollective 的 Puppet Agent 插件还有很多辅助参数，我们可以通过 mco help puppet 查看到它们。

```
    count    返回一个运行中，可用状态与关闭状态的节点统计信息
    enable   - 启动 Puppet Agent 节点
    disable  - 关闭 Puppet Agent 节点
    resource - manage individual resources using the Puppet Type (RAL) system
    runall   - invoke a puppet run on matching nodes, making sure to only run
                   CONCURRENCY nodes at a time
    runonce  - 运行一个 Puppet Agent 节点
    status   - 简要地显示 Puppet Agent 运行状态报告
    summary  - 显示资源与运行的总结
```

以下为 MCollective 的 Puppet Agent 插件，两个 count 与 status 参数的使用方式。

1）查看所有 Puppet Agent 的运行状态。

```
    # mco puppet count
    Total Puppet nodes: 2
            Nodes currently enabled: 2
            Nodes currently disabled: 0
    Nodes currently doing puppet runs: 0
             Nodes currently stopped: 2
          Nodes with daemons started: 0
       Nodes without daemons started: 2
          Daemons started but idling: 0
```

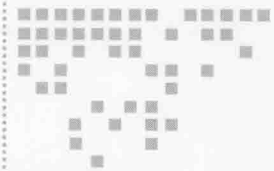
2）简要地显示 Puppet Agent 运行状态报告。

```
    # mco rpc puppet status
    * [ ============================================================> ] 2 / 2
      web.exmaple.com: Currently stopped; last completed run 56 days 8 hours 57 minutes
13 seconds ago
          web.exmaple.com Currently stopped; last completed run 06 seconds ago
    Summary of Applying:
      false = 2
    Summary of Daemon Running:
      stopped = 2
    Summary of Enabled:
      enabled = 2
    Summary of Idling:
      false = 2
    Summary of Status:
      stopped = 2
    Finished processing 2 / 2 hosts in 149.06 ms
```

第四部分 Part 4

应 用 篇

- 第16章 HAProxy构建Puppet集群实践
- 第17章 Puppet管理SSO实践
- 第18章 Puppet快速构建企业内部网实践



Chapter 16 第 16 章

HAProxy 构建 Puppet 集群实践

Puppet 的性能与版本升级一直是被大型网站与企业内部网管理员所关注的两个方面。本章将结合之前章节学习过的知识整合 HAProxy 与 Puppet 来解决管理员日常关注的两个问题。通过本章的学习也能让读者开拓思路，加深对整个 Puppet 的管理与配置过程的了解。本章首先介绍 HAProxy 这款性能优越的负载均衡软件；接着介绍 HAProxy 的初始化过程；然后介绍如何在我们之前搭建过的 Master 基础上扩展 Puppet 集群的流程；最后介绍 Puppet 的升级流程。

16.1 HAProxy 简介

HAProxy 是一款完全免费的高性能负载均衡软件，与费用高昂的硬件负载均衡相比有着良好的性能与成本的优势。HAProxy 的工作流程如图 16-1 所示。网络流量经过防火墙后访问 HAProxy，HAProxy 又将流量平均的分配到后端的 Web Server 达到负载均衡的目的。HAproxy 支持 TCP 和 HTTP 应用层代理，适用于大型网站与企业。

1. HAProxy 的优势

我们曾在第 11 章介绍过一些常见的负载均衡技术，这里单从非商业的负载均衡技术来说，对比 HAProxy 与 Nginx 两款开源软件。HAProxy 有如下的优势。

- HAProxy 规避了 Nginx 的一些缺点，如 Session 的保持与 Cookie 的引导等工作。
- 单纯从效率上来讲 HAProxy 有着更出色的性能优势，在并发处理上也优于 Nginx。
- HAProxy 支持 url 状态检测。

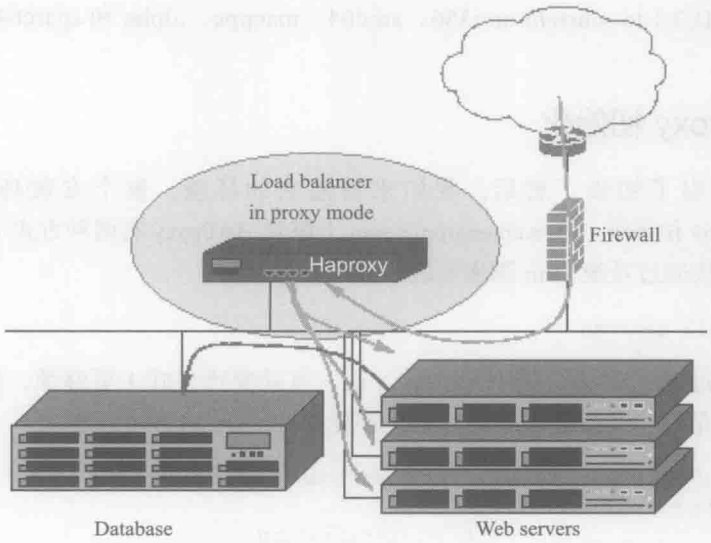


图 16-1 HAProxy 工作流程

2. HAProxy 均衡负载算法

目前 HAProxy 支持以下 8 种负载均衡的算法。

1）roundrobin：轮询；

2）static-rr：根据权重；

3）leastconn：最少连接者先处理；

4）source：表示根据请求源 IP，这个与 Nginx 的 IP_hash 机制类似，我们可以将其作为解决 Session 问题的一种方法；

5）ri：表示根据请求的 URI；

6）rl_param：表示根据请求的 URl 参数；

7）hdr(name)：表示根据 HTTP 请求头来锁定每一次 HTTP 请求；

8）rdp-cookie(name)：表示根据 cookie（name）来锁定，并哈希每一次 TCP 请求。

3. HAProxy 支持的平台

目前 HAProxy 支持的系统与平台如下。

1）Linux 2.4x86_32 位或 x86_64 位、Alpha、SPARC、MIPS 和 PARISC

2）Linux 2.6 x86_32 位或 x86_64 位、ARM (ixp425)、PPC64

3）Solaris 8/9 的 UltraSPARC 2 and 3

4）Solaris 10 的 Opteron and UltraSPARC

5）FreeBSD 4.10 - 8 的 x86 平台

6）OpenBSD 3.1 to -current on i386、amd64、macppc、alpha 和 sparc64

16.2 HAProxy 初始化

对 HAProxy 有了初步了解后，我们来搭建它的环境，整个安装环境我们使用了 CentOS6.5_x86_64 位系统。在 web.example.com 上搭建 HAProxy 有两种方式（推荐方式 2）。

方式 1 直接通过系统 yum 源来安装。如下：

```
# yum install haproxy
```

方式 2 通过 Puppet 来部署 HAProxy，这一方式要比方式 1 更麻烦，但优势是后续在其他系统再次安装时会更加省时省力。首先创建 HAProxy 的目录结构。

```
# mkdir -p /etc/puppet/modules/haproxy/{manifests,templates}
# tree /modules/nginx/manifests/
                          |      |-- init.pp
                          |  __ templates
                                 |-- haproxy.cfg.erb
```

接着编辑 init.pp 文件，追加以下内容到文件中。

```
class haproxy {
 # 安装 HAProxy 软件包
 package { "haproxy":
  ensure => installed,
 }
 # 从 Master 同步配置文件到 HAProxy 服务器
 file { "/etc/haproxy/haproxy.cfg":     # 配置文件目标
  source  => "puppet:///modules/haproxy/haproxy.cfg", # 同步配置源
  require => Package["haproxy"],      # 同步配置前需要确认标题为 HAProxy 的 package 资源已经成功执行
  notify  => Service["haproxy"],      # 成功同步配置文件后，通知启动 HAProxy 守护进程
 }
 # 启动 HAProxy 守护进程
  service { "haproxy":
  ensure  => running,
  enable  => true,
  require => Package["haproxy"],
 }

}
```

haproxy.cfg.erb 为 HAProxy 的主配置文件。我们在 Master 上编辑 HAProxy 的 /modules/nginx/manifests/templates/haproxy.cfg.erb 主配置文件，内容如下：

```
# 全局配置
global
  daemon      # 以后台方式运行 daemon
```

```
  user  haproxy     # 后台运行的账户
  group  haproxy     # 后台运行的组
  maxconn  4000     # 默认最大的连接数
  pidfile  /var/run/haproxy.pid     # HAProxy 的 pid 存放路径
# 默认配置
defaults
  mode http    # 使用的默认协议,HAProxy 默认支持 3 种模式 (http|tcp|health),其中 http 为应
               # 用层协议 ,tcp 为四层协议
  log  global
  maxconn  8000     # 最大连接数
  option  redispatch     # 失败后是否允许重新分配在 Session
  retries  3     # 失败重试 3 次
  stats  enable     # 开启管理界面功能
  timeout  http-request 10s     # 默认 HTTP 的超时时间
  timeout  queue 1m     # 默认队列超时时间 (1 分钟)
  timeout  connect 10s     # 默认连接超时时间 (10 秒)
  timeout  client 1m     # 默认客户端超时时间 (1 分钟)
  timeout  server 1m     # 默认服务器端超时时间 (1 分钟)
  timeout  check 10s     # 心跳检测超时 (10 秒)
# 管理界面配置
listen stats :80     # 管理界面监听端口
      mode http     # 管理界面展示方式
      stats uri /     # 管理界面根目录
      stats auth user:password     # 登录管理界面的账户与密码
# 负载均衡配置
listen puppet 0.0.0.0:8140 # puppet 用于负载均衡的分类名,会在 Web 的管理界面显示其分类标识
  mode tcp     # 轮询协议
  balance  roundrobin     # 调度算法
  option  tcplog  # 开启 tcp 连接跟踪状态
  option  ssl-hello-chk     # 使用 sslv3 对服务器做健康状况检查
      server  puppet 192.168.110.128:8140 check maxconn 100     # real server
      server  puppet 192.168.110.129:8140 check maxconn 100     # real server
```

在 web.example.com 上执行以下命令。

```
# puppet agent --server puppet.example.com --test
info: Caching catalog for web.example.com
info: Applying configuration version '1401607459'
notice: /Stage[main]/Haproxy/Package[haproxy]/ensure: created
notice: /Stage[main]/Haproxy/File[/etc/haproxy/haproxy.cfg]/content:
Service[haproxy]
notice: /Stage[main]/Haproxy/Service[haproxy]/ensure: ensure changed 'stopped' to
'running'
notice: /Stage[main]/Haproxy/Service[haproxy]: Triggered 'refresh' from 1 events
notice: Finished catalog run in 30.18 seconds
```

最后我们通过查看管理界面的方式确认 HAProxy 已经正常工作。在 IE 中输入 web.example.com 或者对应的 IP 后，结果如图 16-2 所示。

图 16-2 HAProxy 管理界面认证

根据 haproxy.cfg.erb 文件中的配置，默认的用户名为 user，密码为 password。我们可以通过修改 haproxy.cfg.erb 配置文件更改默认密码。登录后如图 16-3 所示。

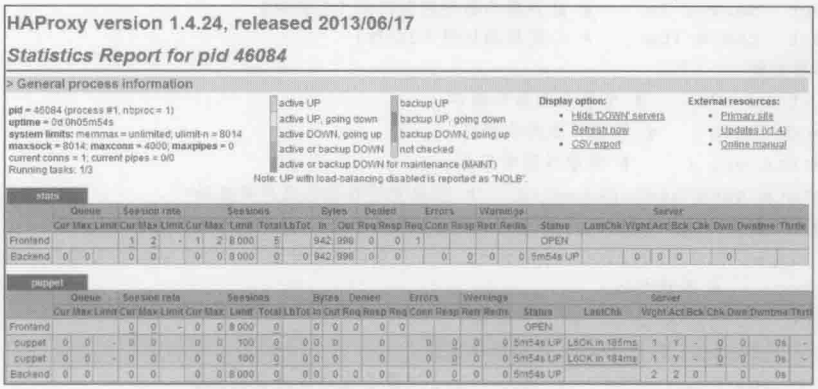


图 16-3 HAProxy 管理界面

16.3 HAProxy 构建 Puppet

本节将在搭建好的 Master 的基础上来介绍如何通过 HAProxy 扩展 Puppet 集群与对 Puppet 版本的升级。首先来介绍 HAProxy 构建 Puppet 的应用场景，某企业年初有 1000 台 PC 在通过 Puppet 配置管理系统获取配置信息，但是由于建立企业之初没有考虑到后续的快速发展和 Puppet 早期版本的缺陷，所以目前企业只有一台单独的服务器，使用的是 Puppet 2.6.18 版本来管理配置所有节点。但随着企业规模扩大，员工队伍也不断壮大，预计到年底员工数在 1000 的基础上还会再增加 3000 人，每人一台 PC，到年底需要再增加 3000 台。另外以后每年员工数都会以 20% 的速度递增，但现状是 Puppet 配置管理系统的处理性能和版本缺陷导致的后果已经显现，Agent 在获取配置信息时 Master 不断超时，导致获取信息失败，员工无法正常工作。系统管理员不得不考虑为配置管理系统加一层负载均衡设备，

在解决性能问题的同时为后续的平滑扩容与版本升级打下基础。最后管理员综合了很多原因，如性能、成本和可扩展性等进行考虑，最终选择了 HAProxy 这款免费的开源软件技术解决方案，来解决日常中遇到的问题。

16.3.1 利用 HAProxy 扩展 Puppet 集群

笔者在搭建好的一台 Master 基础上再扩展 3 台 Puppet Master，并结合 HAProxy 组建 Puppet 集群，Puppet 集群如图 16-4 所示。

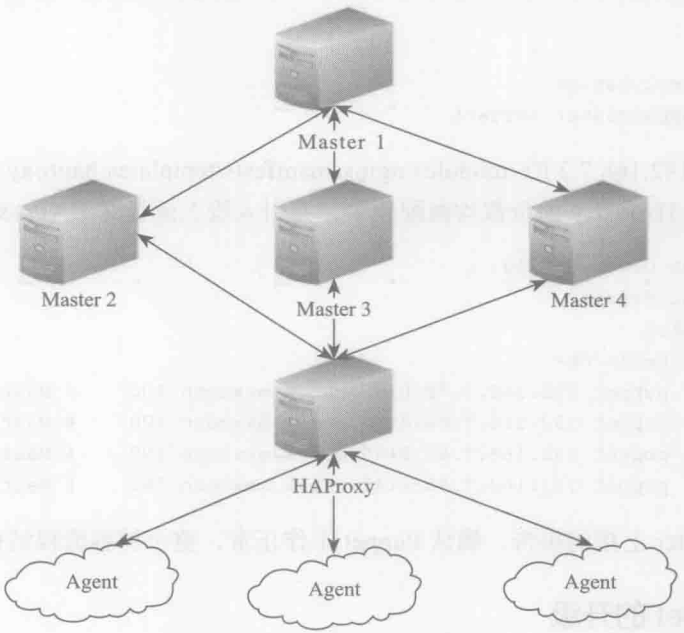


图 16-4 HAProxy 集群架构图

通过图 16-4 可了解架构中的 IP 与机器的现状，机器的部署情况与 IP 如表 16-1 所示。

表 16-1 HAProxy 环境部署表

节 点 名	IP	备 注
Master1	192.168.7.78	使用中的 Master
Master2	192.168.7.86	扩容
Master3	192.168.7.87	扩容
Master4	192.168.7.88	扩容
HAProxy	192.168.7.2	已经部署

我们准备开始扩容，将已经在线上使用 Master(IP: 192.168.7.78) 的证书目录进行打包，并复制到要扩容的服务器上，扩容流程如下：

1）参考第 3 章初始化扩容机器的 Puppet 环境。

2）在已经运行的 Puppet（192.168.7.78）上打包证书目录，默认位置在 /var/lib/puppet/ssl。

```
# tar -cvzf  ssl.tar.gz  /var/lib/puppet/ssl
```

3）将 ssl.tar.gz 证书分别复制到 192.168.7.86、192.168.7.87 和 192.168.7.88 服务器的 /var/lib/puppet 目录下。以 192.168.7.86 为例，执行过程如下：

```
# scp ssl.tar.gz 192.168.7.86:var/lib/puppet
```

4）在 192.168.7.86、192.168.7.87 和 192.168.7.88 服务器上解压 ssl.tar.gz 文件，并重启守护进程。

```
# tar -xvzf ssl.tar.gz
# service puppetmaster restart
```

5）编辑 IP:192.168.7.2 的 /modules/nginx/manifests/templates/haproxy.cfg.erb 配置，将 Master 的 IP 加入 HAProxy 的负载均衡配置中，并引入线上流量对 HAProxy 进行测试。

```
listen puppet 0.0.0.0:8140
 balance  roundrobin
 option  tcplog
 option  ssl-hello-chk
     server  puppet 192.168.7.78:8140 check maxconn 100     # Master1
     server  puppet 192.168.7.86:8140 check maxconn 100     # Master2
     server  puppet 192.168.7.87:8140 check maxconn 100     # Master3
     server  puppet 192.168.7.88:8140 check maxconn 100     # Master4
```

6）观察 Master 上报的报告，确认 Puppet 工作正常，整个扩容流程结束。

16.3.2 Puppet 的升级

在对 Puppet 升级之前，笔者建议先阅读第 2 章的 Puppet 升级注意事项。本节将介绍 Puppet 实际升级的流程。

我们在 HAProxy 集群搭建完成的基础上升级 Puppet 3 的版本，如图 16-5 所示。从架构图中我们了解到，Master 1、Master 2、Master 3 和 Master 4 是已经挂在负载均衡（HAProxy）上提供服务的 4 台 Master 服务器。目前 4 台服务器的操作系统使用了 CentOS 6.5 和 Puppet 2.6.18 版本，由于业务的原因需要扩容一台服务器（Master 5），并将扩容后的服务器升级到 Puppet 3 版本，后续扩容后的服务器（Master 5）运行稳定后，再更新旧服务器（Master 1、Master 2 、Master 3 和 Master 4）的 Puppet 版本。

扩容与升级的流程如下：

1）将 Master 1 上的 Puppet 配置目录（/etc/puppet/）与证书目录（/var/lib/puppet/ssl/）提交到 SVN。

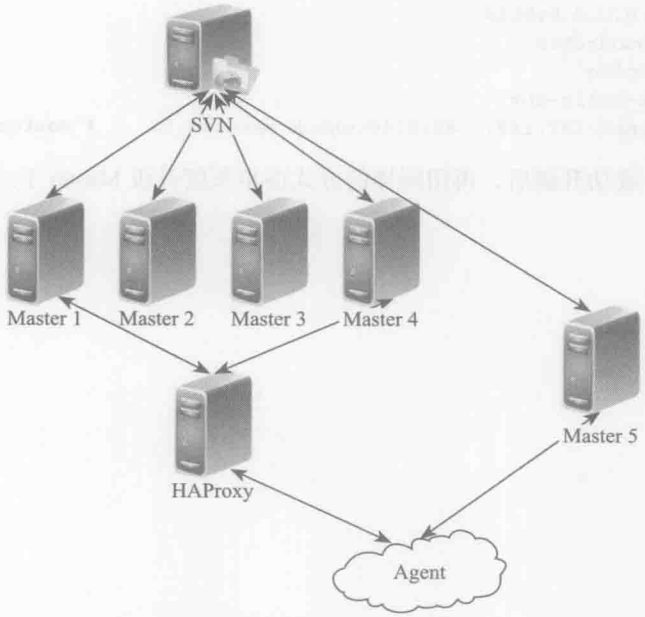


图 16-5 Puppet 架构图

2）将源 Master 1 的 Puppet 的环境复制到 Master 5 上，再将 SVN 上的 Puppet 配置信息 check out 到 Master 5。

3）在 Master 5 上通过 yum update puppet 来升级 Puppet 3，如图 16-6 所示。升级后启动 Master 的守护进程（service puppetmaster start）。

Package	Arch	Version	Repository	Size
Updating:				
puppet	noarch	3.6.1-1.el6	puppetlabs-products	1.3 M
Installing for dependencies:				
hiera	noarch	1.3.3-1.el6	puppetlabs-products	23 k
ruby-rgen	noarch	0.6.5-2.el6	puppetlabs-deps	237 k
rubygem-json	x86_64	1.5.5-1.el6	puppetlabs-deps	763 k
Updating for dependencies:				
facter	x86_64	1:2.0.1-1.el6	puppetlabs-products	84 k
puppet-server	noarch	3.6.1-1.el6	puppetlabs-products	24 k
Transaction Summary				

图 16-6 Puppet 升级

4）将一台 Agent 的 HOST 指向升级后的 Master 5，并观察 Master 5 的 Puppet 日志运行状况。

5）待观察 Puppet 日志没有问题后，将 Master 5 的 IP 加入 HAProxy 中，负载均衡的权重调整为 50，持续观察 Agent 上报的日志是否正常。

```
listen puppet 0.0.0.0:8140
   balance  roundrobin
    option  tcplog
   option  ssl-hello-chk
   server  puppet 192.168.7.86:8140 check maxconn 50    # Master5 权重为 50
```

6）待 Master 5 成功升级后，再用同样的方式逐步灰度升级 Master 1、Master 2、Master 3 和 Master 4。

第 17 章 Chapter 17

Puppet 管理 SSO 实践

多年前笔者曾负责某网站的 SSO 系统的开发与运维工作，在互联网快速发展的年代经历了通过脚本运维管理网站迫使运维人员重复劳动与工作效率低的烦恼。如 SSO 系统模块又多又庞大，每个人负责系统的一部分，有着自己相对独立的一套脚本来运维管理系统模块，模块之间的脚本功能不能被继承与复用，模块文档跟不上加上人员的流动，导致部分系统模块缺乏运维文档，事故频发；再如系统频繁的推送配置，在没有灰度配置文件的情况下变更线上系统导致系统服务异常或不可用等。这些都是我们通过脚本运维经常遇见的问题与烦恼，而这些问题都可以通过 Puppet 配置管理系统来很好地解决。所以本章笔者将自身运维 SSO 系统经历与 Puppet 配置管理系统相结合，让读者能够在模拟真实的系统架构下学习 Puppet 的配置管理系统过程。

17.1 SSO 介绍

17.1.1 什么是 SSO

随着互联网信息时代的发展，各种应用技术层出不穷，每天用户需要穿梭在不同的系统之间，如邮件系统、社交系统、论坛和办公系统等。每个系统都要遵循一定的安全策略，如要求用户按照指定的规则输入用户的 ID 和口令等。随着登录系统的增多，出错的可能性就会增加，受到非法截获和破坏的可能性也会增大，安全性就会相应降低，而如果用户忘记了口令不能执行任务，就需要请求网站管理员的协助，这些既造成了系统和管理资源的开销与浪费同时也降低了生产效率。针对这些问题，通常一些网站或企业会采用 SSO

（Single sign-on，单点登录系统）来解决。简单来说 SSO 就是在一个多系统共存的环境下，用户在一处登录后，就不用在其他系统再次登录的技术解决方案。SSO 优势在于既可以提升用户体验，又可以降低安全风险，因此在很多网站与企业中被广泛使用。

17.1.2 SSO 系统工作流程图

目前很多网站与企业都使用了 SSO 系统，SSO 系统工作流程如图 17-1 所示。

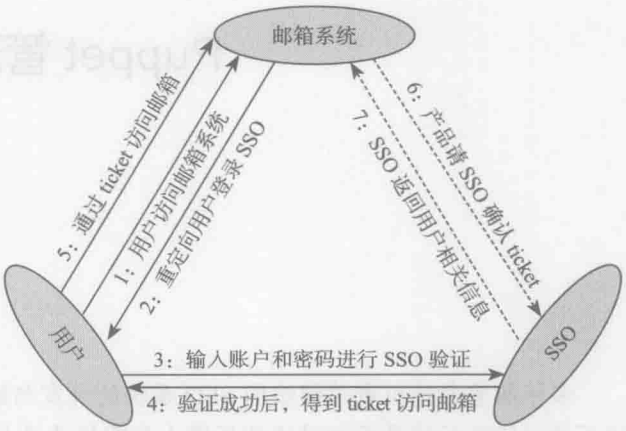


图 17-1 SSO 工作流程图

下面笔者以用户访问邮箱系统为例，具体分析 SSO 的整个工作流程。

1）用户首先访问邮箱系统，选择登录邮箱系统。

2）邮箱系统通过 HTTP 协议将用户重定向到 SSO 系统的登录页面。

3）用户在 SSO 系统的登录页面提交账号和密码，并交由 SSO 系统后端处理。

4）SSO 系统根据用户提交的账号与密码信息进行验证，验证成功后生成用户的 ticket（ticket“票据”并不是用户的密码，它只是一串根据用户信息生成的随机密钥，存放在 SSO 系统中，提供后续的验证使用）。

5）用户带着从 SSO 系统返回的 ticket 去访问邮箱系统，并将 ticket 交给邮箱系统。

6）邮箱系统接收到用户的 ticket 后，再将 ticket 转发给 SSO 系统验证用户的 ticket 信息是否正确。

7）如果验证 ticket 成功，SSO 系统会向邮箱系统返回与用户相关的信息。至此用户登录成功，整个流程结束。

从 SSO 系统工作流程图中可以看到，邮箱系统并没有存放用户的相关信息，特别是比较敏感的账户与密码信息，邮箱系统只通过用户携带的 ticket 从 SSO 系统获取用户信息。当用户从其他系统登录，如社交系统、论坛系统和办公系统等，都可以通过用户携带的 ticket 从 SSO 系统认证后获取用户的信息，自动为用户在本系统登录，从而达到了一次登录，漫游全网的功能。

17.1.3 SSO 系统架构

了解了什么是 SSO 系统和其工作原理后，我们再来看某大型网站的 SSO 系统架构。某大型网站有很多的子站点，如新闻、娱乐、财经、科技、体育和房产等。网站希望通过

SSO 系统对用户进行统一的登录，提高用户的安全性，同时降低产品的维护成本。网站预设每天有 6 亿的 PV 访问量（Page View，页面访问量），8000 万的同时在线用户。为了提升用户的产品体验，SSO 系统作了异地部署，分别在北京网通机房和上海电信机房部署服务，使用户在可就近接入的同时实现了服务的灾备与高可用。这样一套复杂的 SSO 系统架构是什么样的呢？如图 17-2 所示（实际的系统架构要比这复杂很多，但为降低学习的成本，笔者做了简化处理，只标识了一些重要的模块和网络访问关系）。

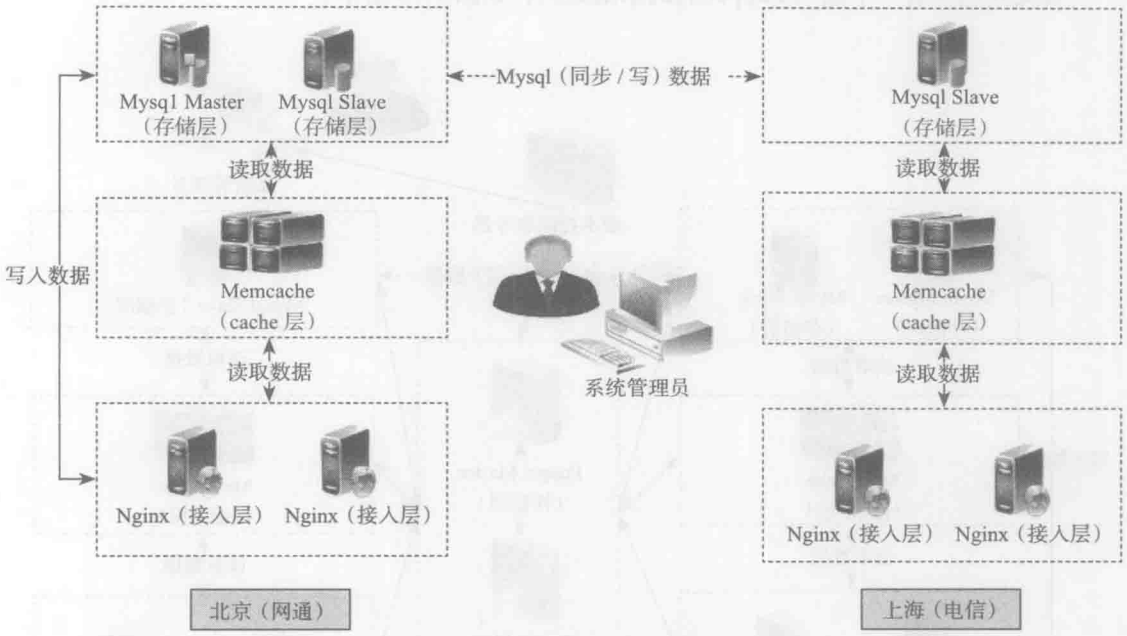


图 17-2 SSO 系统架构图

从图 17-2 中可以看到，SSO 系统主要分为接入层、Cache 层和存储层这 3 层，它们的作用分别如下。

- 接入层：通过 Web 服务器接收用户的请求，并根据请求的逻辑进行判断。如果用户是要注册账户，逻辑直接将用户的信息写入 MySQL 主库。如果用户是通过其账号和密码进行验证身份，则优先验证 Cache 层的用户信息，如果 Cache 层没有相应信息再访问后端的 MySQL 数据库，从 MySQL 数据库验证并获取信息后，将信息返回给用户，再将信息写入 Cache 层。
- Cache 层：用来将账户相关信息缓存在内存中。缓存在内存中的优势是提升用户获取信息的速度，降低对后端数据库的压力，将各层各环节功能发挥到极致，特别是在服务异地部署的情况下，Cache 层的优势比较明显。
- 存储层：存储层存放了与账户相关的信息。通常在大型的系统中，我们会对 MySQL

进行读写分离，读写分离的优势是可以提高 MySQL 的性能与可扩展性。

17.2 通过 Puppet 管理与运营 SSO 系统

目前我们已经对什么是 SSO 系统、SSO 系统的工作流程和架构有了基本的了解。本节将主要介绍通过 Puppet 管理与运营 SSO 系统。

首先我们来看一下加入 Puppet 后的系统架构，如图 17-3 所示。

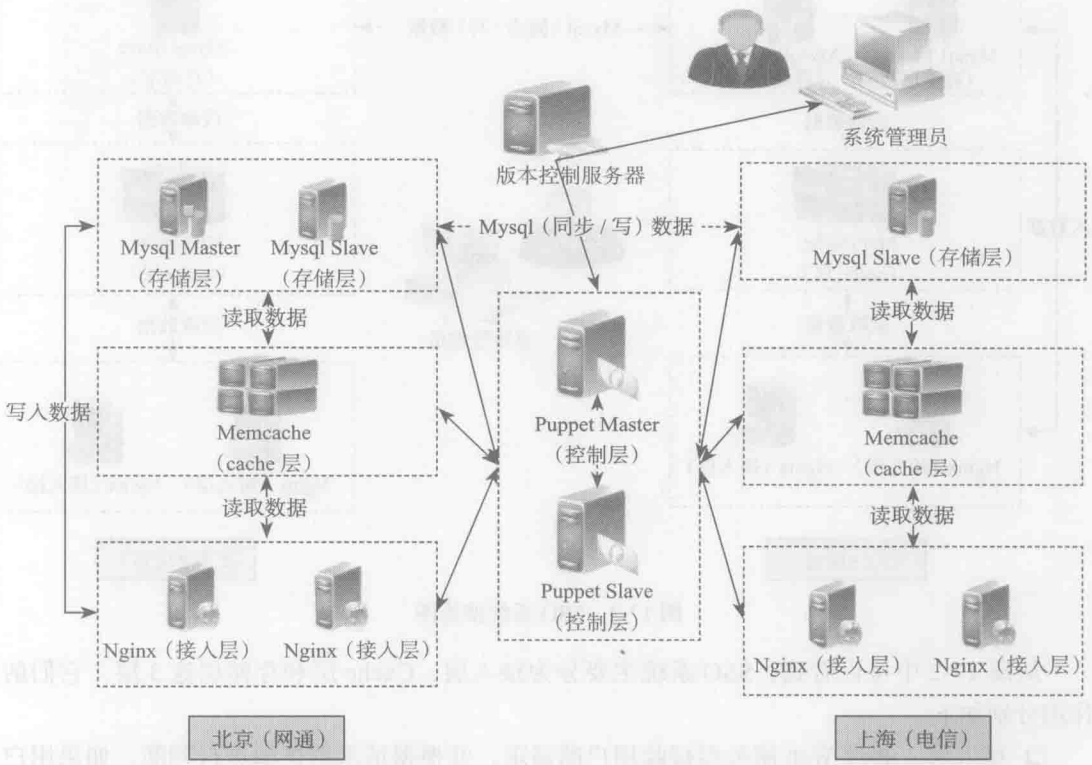


图 17-3 Puppet 运营与管理 SSO 系统

加入了 Puppet 后，新的架构图由之前的 3 层结构演变成了 4 层结构，多了控制层 Puppet。由于跨 IDC 访问的原因，笔者建议这里使用双 Puppet 的部署解决方案，在北京与上海机房分别部署 Puppet Master，让各 IDC 的服务器就近接入 Puppet Master，提升各 IDC 管理配置的效率。

接着根据架构图的演变，我们再来看一下架构图中各层的 Hostname、IP 和操作系统发行版本的情况与建议，本节的环境配置如表 17-1 所示。

❑ Hostname：因为 Puppet 在配置管理过程中主要识别主机的 Hostname，所以建议读

者在设置 Hostname 时要有规划与规则，Hostname 要见名思义。本节的 Hostname 命名规范为 IDC- 架构层 - 产品名 - 组，以“-”进行分隔，4 键为一组来标识一台主机，后接 .example.com，这主要为后续与 ENC 结合考虑。

- IP：如果我们是新建的 SSO 系统，建议每层都要统一网段，最好是连续的 IP，后续 Puppet 代码逻辑中会减少很多的逻辑判断，降低维护成本。如果读者已经将 Puppet 引入了线上环境，那前期免不了需要对各层的 IP 多做一些逻辑判断，后期需要考虑尽量将每层规划到一个 IP 网段进行管理。
- 操作系统发行版本：虽然管理不同发行版本的操作系统是 Puppet 的强项，但是我们也需要考虑 SSO 的程序兼容性，所以建议使用统一操作系统发行版本。本节是基于 CentOS-6.5_x86.64 位系统进行介绍的。

表 17-1 SSO 架构图配置列表

架 构 层	Hostname	IP	操作系统发行版本
存储层（DB）	bj-db-master-1.example.com	10.55.38.40/32	CentOS-6.5_x86.64
	bj-db-savle-1.example.com	10.55.38.41/32	CentOS-6.5_x86.64
	sh-db-savle-1.example.com	10.49.38.40/32	CentOS-6.5_x86.64
缓存层（Cache）	bj-cache-memcache-1.example.com	10.55.38.70/32	CentOS-6.5_x86.64
	sh-cache-memcache-1.example.com	10.49.38.70/32	CentOS-6.5_x86.64
接入层（Web）	bj-access-web-nginx1.example.com	10.55.38.100/32	CentOS-6.5_x86.64
	bj-access-web-nginx2.example.com	10.55.38.101/32	CentOS-6.5_x86.64
	sh-access-web-nginx1.example.com	10.49.38.100/32	CentOS-6.5_x86.64
	sh-access-web-nginx2.example.com	10.49.38.101/32	CentOS-6.5_x86.64
控制层（Puppet）	bj-puppet-master-1.example.com	10.55.38.50/32	CentOS-6.5_x86.64
	sh-puppet-master-1.example.com	10.49.38.50/32	CentOS-6.5_x86.64
版本控制	bj-svn-master-1.example.com	10.55.48.49/32	CentOS-6.5_x86.64

17.2.1 Puppet 系统初始化

首先初始化 SSO 系统的软件环境，分别为 Ruby 和 Puppet。以北京网通机房的 Master（Hostname:bj-puppet-master-1.example.com，对应 IP 为 10.55.38.50）为例，配置流程共分 4 步（除 Puppet Master 以外，Agent 系统初始化只需要关注前 3 步）。

1）安装默认的 yum 和 Puppet 的软件源。

```
# 推荐安装国内的 yum 软件源
wget http://mirrors.163.com/.help/CentOS6-Base-163.repo -P /etc/yum.report.d/
# 安装 Puppet 官方 yum 源
rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-el-6.noarch.rpm
# 导入 GPG 密钥
rpm -import http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
```

2）安装 Ruby 环境。

```
# yum install ruby ruby-libs rubygems
```

Ruby 环境安装完成后，推荐将 gem 更换为国内 gem 源。

```
#ruby.taobao.org
gem sources --remove https://rubygems.org
gem sources -a http://ruby.taobao.org
gem sources -l
```

3）安装 Puppet 与 Facter。

由于不同版本间在使用时有些许差异，为避免版本差异带来异常问题，所以 SSO 系统以 Puppet 2.7.25 稳定版本来做演示。在安装 Puppet 时指定 Puppet-2.7.25 的版本。

```
# yum install puppet-2.7.25 puppet-server-2.5.25 facter
```

4）初始化 Master/Agent 的配置。

Master 配置如下。

Puppet 环境安装后，如果有特殊配置可以参考本书第 4.2 章中的 Puppet 主要文件配置一节，如果无特殊配置可以直接启动 Master 的守护进程。

```
# service puppetmaster start
```

守护进程启动过程中需要注意以下常见的 3 种错误：

- Master/Agent 时间不一致导致 SSL 认证失败。
- Agent 访问 Master 过程中如果报 err: Could not request certificate: No route to host-connect(2) 错误，请确认 8140 端口是否已经开启。开启命令如下：

```
# iptables -t filter -A  INPUT -p tcp -m state -state NEW --dport 8140 -j ACCEPT
```

- 更改 Master 的 Hostname 为 bj-puppet-master-1.example.com，否则有可能会报 err: Could not retrieve catalog from remote server: Server hostname ' bj-puppet-master-1.example.com ' did not match server certificate; expected one of bogon.localdomain, DNS:bogon.localdomain, DNS:puppet, DNS:puppet.localdomain 错误。设置命令如下：

```
# hosntmae bj-puppet-master-1.example.com
```

Agent 配置如下。

在连接前首先设置 /etc/host 和 /etc/host.conf 两个配置文件（或者参考第 3 章通过配置 Dnsmasq 来设置 Master 的域名访问方式）。host 配置文件的作用是用来增加对应域名与 IP 的关系；host.conf 配置文件用来设置 host 与 bind 的解析顺序。

```
# 配置 host , 格式 IP | FQDN | HOSTNMAE
echo  " 10.55.38.50 bj-puppet-master-1.example.com  bj-web-nginx-1.example.com " >>
/etc/hosts
# 配置 host.conf
```

```
echo " order hosts,bind " >> /etc/host.conf
```

设置 Agent 的 Hostname。

```
# hosntmae bj-access-web-nginx1.example.com
```

访问 Master，测试配置是否成功。

```
puppet agent --server puppet.example.com --test
```

17.2.2 Puppet 配置管理环境的初始化

目前我们已经搭建好了整个 Puppet 的 Master/Agent 环境，下面就是通过 Puppet 来管理配置整个 SSO 系统。Master 整个配置过程如下：

- 配置文件初始化。
- Modules 基础模块的接入层（Web）的初始化。
- Modules 基础模块的缓存层（Cache）的初始化。
- Modules 基础模块的存储层（DB）的初始化。
- Manifests 逻辑初始化。

1. 配置文件初始化

Master 配置文件的初始化包含了 puppet.conf、autosign.conf 和 fileserver.conf 等。

1）修改 puppet.conf 配置文件。在运营环境中为了尽量避免配置文件导致的线上故障，笔者建议在通过 Puppet 更新配置前先对配置灰度，所以我们要打开 Puppet 的灰度控制开关，也就是我们在第 5 章中所介绍的“环境”。在 Master（Hostname:bj-puppet-master-1.example.com，IP: 10.55.38.50）上的修改 puppet.conf 主配置文件，增加环境功能。增加配置前先创建环境的配置目录。

```
# mkdir -p /etc/puppet/environments/{production,testing, development}
# mkdir -p /etc/puppet/environments/production/{manifests,modules}
# mkdir -p /etc/puppet/environments/testing/{manifests,modules}
# mkdir -p /etc/puppet/environments/development/{manifests,modules}
```

修改 /etc/puppet/puppet.conf 配置文件。

```
[production]    # 线上环境
manifest = /etc/puppet/environments/production/manifests/site.pp
modulepath = /etc/puppet/environments/production/modules
[development]    # 开发环境
manifest = /etc/puppet/environments/development/manifests/site.pp
modulepath = /etc/puppet/environments/development/modules
[testing]    # 测试环境
```

```
manifest = /etc/puppet/environments/testing/manifests/site.pp
modulepath = /etc/puppet/environments/testing/modules
```

2）修改 autosign.conf 配置文件。在生产环境中，通常我们会开启 Master 自动授权 Agent 的证书签名功能（此种方式是在读者已经了解了系统架构和网络状况的情况下开启），因为大量的 Agent 首次访问需要授权认证签名，手动方式处理会影响工作效率，所以需要开启 autosign.conf 配置文件功能，提升工作效率。创建 autosign.conf 配置文件，并追加“*”到 autosign.conf 配置文件中。

```
echo "*" > /etc/puppet/autosign.conf
```

3）修改 fileserver.conf 配置文件。它是 Puppet 的 Fileserver，通常我们同步一些比较大的配置文件或软件包时使用 Fileserver。修改 fileserver.conf 配置文件，配置内容如下：

```
[files]
path /data/puppet_file/     # 在 Master 指定文件存放的目录
allow *     # 准许所有机器的请求
```

由于整套 SSO 系统包含了很多软件包与配置文件，所以笔者建议将配置文件目录指定到比较大的磁盘，方便存储更多的配置文件与软件包，并准许局域网内的任何机器访问这些配置文件。

在修改 puppet.conf、autosign.conf 和 fileserver.conf 配置文件后要重启 Master 的守护进程。

2. 接入层（Web）初始化

SSO 系统接入层的作用是接收各产品的用户认证请求并转发到后端存储进行验证。在接入层比较常用的 Web 软件有 Apache、lighttpd 和 Nginx 等。本节以 Nginx 为例来介绍 SSO 系统接入层架构，因为它的并发处理性能与其他软件相比会更强，也有很强的扩展性。首先我们来创建 Nginx 模块的目录与文件结构。

```
# mkdir -p /etc/puppet/environments/production/modules/nginx/
{manifests,templates}
# tree production/modules/nginx/manifests/
                                |     |-- params.pp
                                |     |-- package.pp
                                |     |-- service.pp
                                |     |-- config.pp
                                |     |-- init.pp
                                | __templates
                                      |-- nginx.conf.erb
```

接着来看每个配置文件的内容。

1）params.pp 用于设置 Nginx 的参数列表。

```
class nginx::params{
  $ng_work_conf_dir = "/etc/nginx/"     # 设置 Nginx 根目录
  $ng_work_connections  = "1000"      # 设置 Nginx 连接数
  $ng_port = "80"     # 设置 Nginx 访问端口
  $ng_fqdn = "sso.example.com"     # 设置 Nginx 虚拟机域名
}
```

2）package.pp 用于安装 Nginx 软件包。

```
class nginx::package {
 $packages_array = ['nginx', 'gd',]
case $::operatingsystem {
 centos,fedora,rhel,redhat,centos,scientific:{
  # 由于 RedHat 系列发行版本系统中不包含 Nginx 源，所以需要我们通过 yumrepo 资源来指定 Nginx 源
  yumrepo { "yum_nginx-release":     # 指定 yumrepo 的标题，它后续也是源的文件名
   # $baseurl 为 Nginx 源地址变量，其中地址包含 "$" 需要通过 "\" 进行转义
    baseurl  => "http://nginx.org/packages/centos/6/\$basearch/",
    descr    => 'nginx repo',
    enabled  => '1',
    gpgcheck => '0',
  }
  # 通过 package 安装软件包数组
  package { "nginx":
   ensure  => present,
   # Yumrepo 为公有属性（注意首字母大写），公有属性的作用是在确定 yumrepo 资源执行成功的情况下再
   # 安装软件包
require => Yumrepo['yum_nginx-release'],
  }
}debian,ubuntu: { # 如果是 ubuntu 发行版本，通过 package 资源直接安装软件包数据组
  package { "$packages_array":
  ensure  => present,
  }
}default: {
      notify {"error":}
}
}
}
```

3）config.pp 用于从 Master 同步 Nginx 的配置文件到 Agent。

```
class nginx::config(
 $worker_processes    = $::processorcount,     # 通过 Facter 收集 Agent 硬件信息设置 CPU
的核数
 $worker_connections = $nginx::params::ng_work_connections,     # 赋值 Nginx 连接数
 $worker_dir= $nginx::params::ng_work_dir,     # 赋值 Nginx 根目录
 $worker_port= $nginx::params::ng_port,        # 赋值 Nginx 端口
 $worker_fqdn= $nginx::params::ng_fqdn,        # 赋值 Nginx 的虚拟主机域名
) inherits nginx::params {
 # 初始化 file 资源的权限与用户组
```

```
    File {
     owner => 'root',
     group => 'root',
     mode  => '0644',  }
   # 通过 file 资源中插入 template 函数，同步 Nginx 的配置文件
   file { "${nginx::params::ng_work_conf_dir}/nginx.conf":
     ensure  => file,
     content => template('nginx/nginx.conf.erb'),
    }
   }
```

以下为 Nginx 的模板配置文件，Nginx 配置模板文件存放在 Master 的 /production/modules/nginx/templates/nginx.conf.erb，读者只需关注赋值的内容与方法。

```
   # begin
   worker_processes   <%= worker_processes %>;      # 设置 CPU 核数，通常采用 Facter 收集 Agent 值设置
   events {
      worker_connections  <%= worker_connections %>;   # 设置 Nginx 的连接数
   }
   http {
      include       mime.types;
      default_type  application/octet-stream;
      sendfile        on;
      keepalive_timeout  65;
      server {
         listen       <%= worker_port %>;     # 设置 Nginx 的监听端口
         server_name  <%= worker_fqdn %>;   # 设置 Nginx 的虚拟主机的域名
         #charset koi8-r;
         #access_log  logs/host.access.log  main;
         location / {
            root   html;
            index  index.html index.htm;
         }
         error_page   500 502 503 504  /50x.html;
         location = /50x.html {
            root   html;
         }
   }
   }
   # end
```

4）service.pp 用于启动 Nginx 的守护进程。

```
class nginx::service{
 # 启动 nginx 守护进程
 service { "nginx":
  ensure     => running,
  enable     => true,
```

```
    hasstatus  => true,
    hasrestart => true
  }
}
```

5）init.pp 用于整个 Nginx 的安装、配置与启动逻辑的整合。

```
class nginx (
) inherits nginx::params {
class { 'nginx::package':}
class { 'nginx::config':
  # 同步配置文件后通知 nginx:server 类重新加载配置文件，注意 Class 首字母大写
  notify => Class['nginx::service'],
}
class {'nginx::service':}
}
```

6）Nginx 模块配置好后，在 site.pp 中加载 Nginx 的模块，测试是否正常工作。

```
node default{
 include nginx
}
```

通过（Hostname:bj-access-web-nginx1.example.com，IP:10.55.38.100）的接入层主机测试整个配置流程是否正常。Agent 命令与输出如下：

```
# puppet agent --server bj-puppet-master-1.example.com --test
```

在 Agent 上通过 netstat 命令来确认 Nginx 是否安装成功。

```
netstat -tnl | grep 80
tcp        0      0 0.0.0.0:80              0.0.0.0:*                   LISTEN
```

3. 缓存层（Cache）初始化

在 SSO 的系统架构中，缓存层的作用是将经常访问的用户信息以 key/values 形式缓存在内存中，提高用户访问数据的速度，降低后端数据库的压力。在缓存层，我们使用的是开源软件 Memcached，它是一套分布式的高速缓存系统，由 LiveJournal 的 Brad Fitzpatrick 开发，目前被许多网站使用。Memcached 安装与配置非常简单，它支持 UNIX/Linux 系列操作系统，同时也支持微软的 Windows 系列操作系统。安装后可以通过以下命令启动 Mecached 的守护进程。

```
# memcached -d -m 500 -u root  -p 12000 -c 256
```

如以上参数表示，-d 指以守护进程方式启动；-m 指定使用内存的大小；-u 指定启动的用户；-p 指定 Mecached 以 Socket 方式启动在 12000 端口的监听；-c 指定最大接收并发数。除了上边这些参数外，Memcached 还有其他参数，如表 17-2 所示。

表 17-2 Memcache 其他参数列表

参 数	含 义	参 数	含 义
-p <num>	TCP 监听端口（默认为 11211 端口）	-m <num>	最大的内存使用（默认 64 MB）
-U <num>	UDP 监听端口（默认为 11211 端口）	-M	内存耗尽返回错误
-s <file>	UNIX socket 监听路径，不支持网络	-c <num>	最大并发连接数（默认为 1023）
-a <mask>	UNIX socket 访问掩码，八进制（默认为 0700）	-k	锁定所有分页内存
-l <IP>	监听的服务器 IP 地址（默认为 0.0.0.0）	-v	输出警告和错误信息
-d	守护进程模式启动	-vv	同时打印客户端请求和返回信息
-r	最大限度利用核心文件限制	-vvv	打印内部状态转换信息
-u <username>	运行 memcached 的用户	-i	打印 memcached 和 libevent 版本信息
-P <file>	设置保存 pid 文件	-f <factor>	块大小增长倍数（默认为 1.25）
-n <bytes>	key+value+flags 最小分配空间（默认为 48）	-L	如果有效，尝试使用大内存页。增加内存页大小可以减少失误的 TLB 数量，提高性能
-D <char>	指定 key 和 IDs 的分隔符 default is ":" (colon). 如果指定此选项，统计信息收集自动开启	-t <num>	使用的线程数量（默认为 4）
-R	每个事件的最大请求数（默认为 20）	-C	禁止使用 CAS
-b	设置积压队列数限制（默认为 1024）	-B	绑定协议 - one of ascii、binary or auto（默认）
-I	分配给每个 slab 页（默认为 1MB，最小 1KB，最大 128MB）		

下面通过 Puppet 来配置 Memcached 模块。

1）创建如下目录结构。

```
# mkdir -p /etc/puppet/environments/production/memcached/{manifests,templates}
# tree production/modules/memcached/manifests/
|-- params.pp
|-- package.pp
|-- service.pp
|-- init.pp
```

2）通过 params.pp 定义 Memcached 的启动参数。

```
class memcached::params{
 $mem = "256"     # 使用内存大小
 $user = "root"     # 启动守护进程的账户
 $port = "11200"     # 启动守护进程的端口
 $connection = "1000"     # 连接数
}
```

3）通过 package.pp 安装 Memcached 软件包。

```
class memcached::package{
# 安装 memcached
```

```
    package { "memcached":
      ensure  => present,
    }
  }
```

4）通过 server.pp 启动 Memcached 守护进程。这里启动分为两种情况，一种是默认启动 Memcached 守护进程，它会将端口监听在 11211 上。另一种是自定义启动 Memcached，通过 exec 资源可以自定义设置启动参数，自定期启动参数存放在 parms.pp 文件中。先来看默认启动方式。

```
  class memcached::service{
    # 启动 memcached 守护进程
    service { "memcached":
      ensure     => running,
      enable     => true,
      hasstatus  => true,
      hasrestart => true
    }
  }
```

所用 Agent 机器型号不一致，内存大小自然也不相同。在启动 Memcached 守护进程过程中，不同型号的机器采用相同的内存大小显然不合理的，这时可以通过 Facter 来收集 Agent 的内存信息，并计算每台机器内存剩余容量，然后用剩余容量乘以 50% 将所得值赋给 Memcached 守护进程供其使用，这就是自定义启动方式（为什么要取 50%？其实用户可以随意调整这个百分比，但是建议在 50% 左右或以下，因为操作系统需要一定富余的内存，如果过高可能会导致操作系统异常）。

```
  # 为了方便内存间差值的数学运算，通过 regsubst 函数将 Facter 内存变量中的 GB 单位替换为空
  $mem_free= regsubst($memoryfree," GB","")
  # 计算 50% 内存大小
  $mem = $mem_free * 0.5
  # 通过 exec 资源启动自定义参数的 Memcached 守护进程
  exec { 'memcached':
    command      => "/usr/bin/memcached -d -m $mem -u $memcache::params::user -p
$memcache::params::port -c $memcache::params::connection",
    refreshonly => true,
  }
```

5）通过 init.pp 对整个 Memcached 进行安装、配置与启动逻辑的整合。

```
  class memcached(
  )inherits memcached::params{
    # 安装 Memcached 软件包
    class {'memcached::package':}
    # 启动 Memcached 守护进程
    class { 'memcached::service':
    }
  }
```

6）Memcached 模块配置完成后，在 site.pp 中加载 Memcached 模块，测试是否正常工作。

```
# puppet agent --server=bj-puppet-master-1.example.com --test
```

7）在 Agent 上通过 netstat 命令来确认 Memcached 是否安装成功。

```
netstat -tnl | grep 11211
t        0      0 0.0.0.0:11211           0.0.0.0:*               LISTEN
```

4. 数据层（DB）初始化

存储层我们使用了 MySQL 关系型数据库来存储 SSO 系统相关信息。为了支持更多的访问量，通常使用 MySQL 主 / 从来提供高性能、扩展性与高可用性。下面来创建 MySQL 主 / 从模块的配置目录结构。

```
# mkdir -p /etc/puppet/environments/production/modules/nginx/{manifests,templates}
# tree production/modules/mysqld/manifests/
                                |     |-- params.pp
                                |     |-- package.pp
                                |     |-- service.pp
                                |     |-- config.pp
                                |     |-- init.pp
                                | __templates
                                      |-- master.conf.erb
                                      |-- slave.conf.erb
```

1）通过 params.pp 定义 MySQL 的自定义启动参数。

```
class mysqld::params{
 $master_conf_dir = "/etc/my.cnf"      # 配置文件根目录
 # 主的 my.cnf 配置
 $master_server_id = "1"    # MySQL 主 / 从 ID
 $master_bin_log = "mysql-bin.log"     # MySQL 的 binlog
 $master_db_sso = "sso"      # 同步的数据库名
 $master_ip = "10.55.38.40" # Mysql 主 / 从模式中只能有一个服务器作为主，此处设置主服务器 IP
 # 从的 my.cnf 配置
 $slave_id = "2" # MySQL 主 / 从 id（建议大于 MySQL 在一主多从情况下，通过哈希设置 IP 对应的 ID）
 $master_user = "root"     # 设置从访问 Master 的账户
 $master_password = "root"     # 设置从访问 Master 的密码
 $master_port = "3306"     # 设置从访问 Master 的端口
 $master_connect_retry = "10"     # 设置从访问 Master 的重试次数
 $master_do_db = "sso"     # 设置从同步主的库名
}
```

2）通过 package.pp 安装 MySQL 的软件包。

```
class mysqld::package{
  package { "mysql-server":
```

```
    ensure  => present,
 }
}
```

3）通过 config.pp 配置 MySQL 主 / 从的软件包。

```
#  MySQL 主配置同步
class mysqld::config::master(
  $log_bin = $mysqld::params::master_bin_log,     # 设置 binlog
  $server_id = $mysqld::params::master_server_id,     # 设置 serverid
  $master_db_sso= $mysqld::params::master_db_sso,     # 设置同步库名
) inherits nginx::params {
 # 设置配置文件权限
File {
   owner => 'root',
   group => 'root',
   mode  => '0644',  }
# 同步主的 my.conf 配置文件
file{ "${mysqld::params::master_conf_dir}":
  ensure  => file,
  content => template('mysqld/master.cnf.erb'),
}
}
#  MySQL 从配置同步
class mysqld::config::slave(
 $slave_id = $mysqld::params::slave_id,
 $master_host = $mysqld::params::master_ip,
 $master_user = $mysqld::params::master_user,
 $master_password = $mysqld::params::master_password,
 $master_port= $mysqld::params::master_port,
 $master_connect_retry= $mysqld::params::master_connect_retry,
 $master_do_db = $mysqld::params::master_do_db,
) inherits nginx::params {
  File {
   owner => 'root',
   group => 'root',
   mode  => '0644',  }
file{ "${mysqld::params::master_conf_dir}":
  ensure  => file,
  content => template('mysqld/slave.cnf.erb'),
}
}
```

4）通过 service.pp 启动 MySQL 守护进程。

```
class mysqld::service{
       service { "mysqld":
              ensure     => running,
              enable     => true,
              hasstatus  => true,
              hasrestart => true
       }
}
```

5）通过 init.pp 对整个 MySQL 进行安装、配置与启动逻辑的整合。

```
class mysqld(
)inherits mysqld::params{
# MySQL 安装
class {'mysqld::package':}
# 判断 MySQL 主 / 从 IP，并根据 IP 同步配置文件
if $ipaddress == $mysqld::params::master_ip{
 class {'mysqld::config::master':
      notify => Class['mysqld::service'],
 }
# 首次安装配置 MySQL 后设置账户的密码
# exec{'/usr/bin/mysqladmin -uroot password "root"':}
}else{
 class {'mysqld::config::slave':
      notify => Class['mysqld::service'],
}
# exec{'/usr/bin/mysqladmin -uroot password "root"':}
}
 class {'mysqld::service':}
}
```

6）MySQL 模块配置完成后，在 site.pp 中加载 MySQL 模块，并通过 Agent 的主 MySQL(Hostname: bj-db-master-1.example.com，IP 10.55.38.40）和（Hostname:bj-db-savle-1.example.com，IP10.55.38.41）测试是否正常工作。

```
# puppet agent --server=bj-puppet-master-1.example.com --test
```

7）在 Agent 上通过 netstat 命令来确认 MySQL 主 / 从是否安装成功。

```
netstat -tnl | grep 3306
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN
```

5. Manifests 逻辑初始化

我们已经创建了 SSO 架构的基础层、缓存层和数据层，它们都是 Modules 基础模块的分支。本节将通过 Manifests 来整合整个 Modules 基础模块的逻辑。整合逻辑包含以下几个部分：

- site.pp 文件初始化。
- 各逻辑层创建 user_00 业务账号。
- 各逻辑层就近解析。
- 发布前先灰度，再上线。

1）site.pp 文件中包含了整个 SSO 系统的业务逻辑，逻辑中包含了不同的 IDC 应该匹配哪些节点的配置、哪些是节点的共有配置、哪些是节点的私有配置、当匹配不到节点时的异常情况应该如何处理等。整个逻辑，笔者使用了 Puppet 清单方式来管理，当然读者也

可以选择之前我们掌握的ENC方式，也是非常方便的，这里不再次介绍。编辑site.pp清单，具体如下：

```
# 继承基础模块节点
node base{
  notify {"base":}
  include common
  include route
}
# 逻辑层节点策略配置，匹配来自北京或者上海的Web服务器
node /^(bj|sh)-web-nginx-\d\.example\.com$/ inherits base{
   notify {"from: *-web-nginx-* ":}
  ·include nginx
}
# 缓存层节点策略配置，匹配来自北京或者上海的Cache服务器
node /^(bj|sh)-cache-memcache-\d\.example\.com$/ inherits base{
  notify {"from: *-cache-memcache-* ":}
  include memcached
}
# 存储层节点策略配置，匹配来自北京或者上海的DB服务器
node /^(bj|sh)-db-master-\d\.example\.com$/ inherits base{
  notify {"from: *-db-mysql-* ":}
  include mysqld
}
# 默认节点策略配置
node default{
      notify{"default not mastch":}
}
```

2）创建urer_00业务账号的配置。

```
# cat /etc/puppet/environments/production/common/manifest/common.pp
class common{
user{ "user_00":
 uid => "501",
 gid => "501",
}
}
```

3）就近解析功能。我们可以通过DNS来实现也可以通过Puppet对IP范围进行判断，根据判断的结果指定Host的方式来实现最终的就近解析功能。这里笔者通过host资源指定Host的方式达到就近解析功能。

```
# cat /etc/puppet/environments/production/route/manifest/route.pp
class  route{
  # 匹配来自北京的Agent，设置访问Master为北京的IP
 if $ipaddress =~ /10\.38\.55\.\d/ {
  host { ' bj-puppet-master-1.example.com ':
   ip => '10.55.38.50',
    ensure => present,
```

```
  }
# 匹配来自上海的 Agent，设置访问 Master 为上海的 IP
}elsif $ipaddress =~ /10\.49\.55\.\d/{
 host { ' sh-puppet-master-1.example.com ':
  ip => '10.49.38.50',
    ensure => present,
  }
}else{
  # 默认策略，设置没有匹配到的 IP 访问北京的 Master
  notify{"default ip not match"}
  host { ' bj-puppet-master-1.example.com ':
   ip => '10.55.38.50',
    ensure => present,
  }
 }
}
```

目前我们已经为 SSO 系统完成了一整套的 manifests 和 modules 的配置，配置包括了 Web 层、Cache 层和 DB 层的清单配置功能。最后我们将这一整套的清单配置目录复制到 development（开发环境）和 testing（测试环境）目录中，让整套 SSO 系统实现灰度的功能。SSO 最终的工作流程如图 17-4 所示。

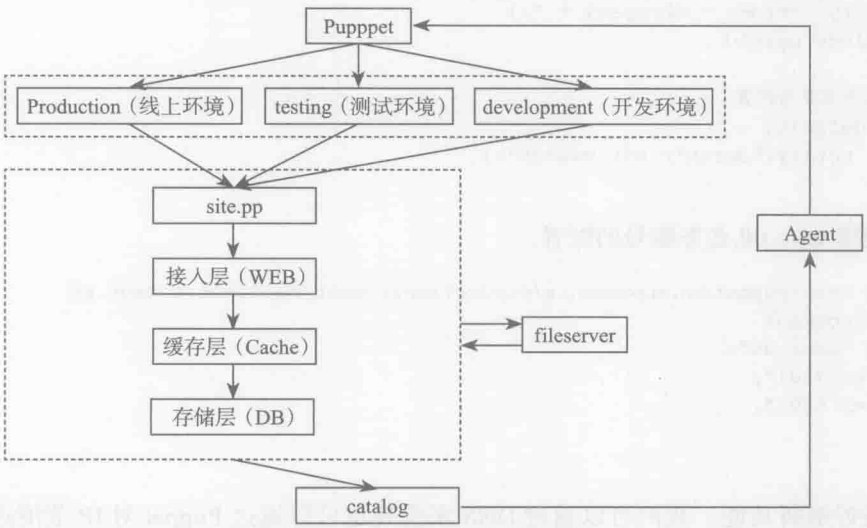


图 17-4 Puppet 环境访问流程图

在 Agent 上通过 --environment 参数来访问 development 和 testing。

```
    # puppet --server bj-puppet-master-1.example.com --environment=development --test
访问开发环境
    # puppet --server bj-puppet-master-1.example.com --environment=testing --test  访
问测试环境
    # puppet --server bj-puppet-master-1.example.com --test  不加 environment 参数，默认
访问线上环境（production）
```

第 18 章 Chapter 18

Puppet 快速构建企业内部网实践

读到这里想必大家对 Puppet 已经有了深入的了解，不过到目前为止我们仍然是以独立的方式重复地开发 Puppet 的代码并应用到线上系统，还是没有真正掌握 Puppet 最精华的部分。其实系统的最终目标就是自动化运维，而要想实现这个目标就需要靠类似 Puppet 这样的工具来协助我们。自动化运维与工具两者相辅相成，工具可以使我们从重复劳动中解脱出来，将更多的精力放在对系统优化与产品优化上，去做更有价值的事情。Puppet 正好为我们提供了这样一系列的辅助工具，帮我们完成重复的工作，这些辅助工具就是基于 Puppet 的 Puppet Forge 与 Example42。Puppet Forge 是一款开源的基础模块免费仓库，由 Puppet 官方提供平台，所有的 Puppet 开发爱好者可以上传基础模块代码，并分享给他人使用。笔者曾在第 5 章提及 Puppet Forge，为了本章案例介绍的承接，本章中笔者将再介绍一遍 Puppet Forge 的基本使用。然后介绍 Example42（官方网站为 www.example42.com，下称 Example42）。Example42 与 Puppet Forge 一样，也是一款开源的 Puppet 基础模块仓库，它的优势在于即装即用，不需独立或重复的开发 Puppet 基础模块代码。Example42 已经帮我们开发好了基础模块的代码，并充分考虑了性能与各操作系统发行版本下的应用场景，在实现系统自动化运维的同时为系统管理员节约了大量的时间成本。本章首先从 Puppet 初始化开始介绍，因为通常企业内部网禁止访问外网，所以本章通过源码方式编译安装 Puppet；接着介绍 Puppet 的辅助工具 Example42 与 Puppet Forge，看它们是如何安装与应用的；最后结合 Puppet 初始化与辅助工具来介绍如何快速构建企业内部网。

18.1 Puppet 初始化

笔者曾在第 3 章介绍过 Puppet 的安装过程，为了文章的完整性，本节笔者再次介绍

Puppet的源码编译安装。本节安装使用Puppet 3.6.2版本作介绍案例，其实本书案例多以Puppet 2.7.25作为案例演示，笔者希望在掌握Puppet 2.7的基础上，也能更多的了解Puppet 3版本知识与内容，它们使用方式与工作原理是一样的，但Puppet 3引入了一些新的功能，Puppet 3.6.2与第3章的Puppet 2.7.25安装也有一些细微的区别，需要读者注意。笔者将Puppet 3.6.2的安装分为以下5步。

1）安装Ruby。同以往一样，我们使用Puppet中兼容性比较好的Ruby1.8.7版本，安装步骤如下。

```
# 下载Ruby1.8.7版本
#  wget  http://ftp.ruby-lang.org/pub/ruby/ruby-1.8.7-p358.zip
# unzip ruby1.8.7
#  cd ruby1.8.7
# 配置Ruby的安装目录
# ./configure --prefix=/usr/local/puppet
# 编译与安装
#  make && make install
# 通过export命令导入ruby命令所在的系统目录
#  export PATH=$PATH:/usr/local/puppet/bin/:/usr/local/puppet/sbin/
# Puppet官方推荐安装以下辅助软件包，辅助软件包通常是Puppet运行或使用时的库文件
#  ruby -r base64 -e "puts:installed"
#  ruby -r cgi -e "puts:installed"
#  ruby -r digest/md5 -e "puts:installed"
#  ruby -r etc -e "puts:installed"
#  ruby -r fileutils -e "puts:installed"
#  ruby -r ipaddr -e "puts:installed"
#  ruby -r openssl -e "puts:installed"
#  ruby -r strscan -e "puts:installed"
#  ruby -r syslog -e "puts:installed"
#  ruby -r uri -e "puts:installed"
#  ruby -r webrick -e "puts:installed"
#  ruby -r webrick/https -e "puts:installed"
#  ruby -r xmlrpc/client -e "puts:installed"
```

在通过源码安装附加软件包过程中，ruby -r webrick/https -e经常会报openssl (LoadError)的错误。

```
# ruby -r webrick/https -e "puts:installed"
/usr/local/services/puppet/lib/ruby/1.8/webrick/ssl.rb:9:in `require': no such file to load -- openssl (LoadError)
```

这一错误是由于我们没有安装openssl软件包所导致的。Puppet在连接过程中使用了SSL协议，SSL协议会用到openssl软件包。如果没有在机器上安装openssl包，可以通过以下方式安装：

```
# wget http://www.openssl.org/source/openssl-1.0.1h.tar.gz
# tar -xvzf openssl-1.0.1h.tar.gz
# cd openssl-1.0.1h && ./confile -fPIC && make && make install
```

安装 openssl 后，再次进入 Ruby 源码包的 ext 目录，安装 Ruby 的 openssl 扩展。方法如下：

```
# cd ruby/ext/openssl/
# ruby extconf.rb --with-openssl-include=/usr/local/ssl/include/ --with-openssl-lib=/usr/local/ssl/lib
# make && make install
```

成功 openssl 扩展后，再次安装 https 包，报错的信息消失。

```
#  ruby -r webrick/https -e "puts:installed"
```

2）下载并安装 Facter。

```
# wget http://downloads.puppetlabs.com/facter/facter-2.0.1.tar.gz
# tar -xvzf facter-2.0.1.tar.gz
# cd facter-2.0.1 && ruby install.rb
```

3）与 Puppet 2.7 的安装有一些区别，Puppet 3 安装时需要预先安装 Hiera，否则会报错。下载 Hiera（下载地址：http://downloads.puppetlabs.com/hiera/）并安装它。

```
# wget http://downloads.puppetlabs.com/hiera/hiera-1.3.4.tar.gz
# tar -xvzf hiera-1.3.4.tar.gz
# cd hiera && ruby install.rb
```

Hiera 是一个“键 / 值”的查询工具，安装它的目的是更好地配置节点，避免做重复的工作。关于 Hiera 的更多信息可以参考官方网站 http://docs.puppetlabs.com/#hierahiera1。

4）下载并安装 Puppet 3.6.2 版本。

```
# wget http://downloads.puppetlabs.com/puppet/puppet-3.6.2.tar.gz
# tar -xvzf puppet-3.6.2.tar.gz
# cd puppet-3.6.2 && ruby install.rb --full
```

安装 Puppet 后，不要急于删除 Puppet 的源码目录，因为我们可以在源码目录中找到在不同发行版本下的配置文件模板，如 puppet.conf、auth.conf、tagmail.conf 和 fileserver.conf 文件等，可以通过 find 系统命令找到它们。

```
# find puppet-3.6.2/ -name "*.conf"
./examples/hiera/etc/puppet.conf
./conf/tagmail.conf
./conf/fileserver.conf
./conf/auth.conf
./lib/puppet/util/libuser.conf
./ext/ips/puppet.conf
./ext/gentoo/puppet/fileserver.conf
./ext/gentoo/puppet/puppet.conf
./ext/debian/fileserver.conf
./ext/debian/puppet.conf
```

```
./ext/redhat/fileserver.conf
./ext/redhat/puppet.conf
./ext/rack/example-passenger-vhost.conf
./spec/fixtures/unit/reports/tagmail/tagmail_passers.conf
./spec/fixtures/unit/reports/tagmail/tagmail_email.conf
./spec/fixtures/unit/reports/tagmail/tagmail_failers.conf
```

这些配置文件的作用已经在第4章详细介绍过，这里不再介绍。我们将puppet.conf、auth.conf、tagmail.conf和fileserver.conf配置文件复制到Puppet配置目录（默认/etc/puppet/）下。配置方法如下：

```
# cp ./conf/tagmail.conf /etc/puppet/     # Puppet邮件发生配置文件
# cp ./conf/fileserver.conf /etc/puppet/     # Puppet文件服务器配置文件
# cp ./conf/auth.conf /etc/puppet/     # Puppet认证配置文件
# cp ./ext/redhat/puppet.conf /etc/puppet/     # Puppet主配置文件
```

5）Master的基础配置。首先在/etc/puppet/puppet.conf中追加Puppet的环境配置，追加方法如下：

```
[production]     # 线上环境
manifest = /etc/puppet/environments/production/manifests/site.pp
modulepath = /etc/puppet/environments/production/modules
[development]     # 开发环境
manifest = /etc/puppet/environments/development/manifests/site.pp
modulepath = /etc/puppet/environments/development/modules
[testing]     # 测试环境
manifest = /etc/puppet/environments/testing/manifests/site.pp
modulepath = /etc/puppet/environments/testing/modules
```

创建这些环境的目录。

```
# mkdir -p /etc/puppet/environments/{production,testing, development}
# mkdir -p /etc/puppet/environments/production/{manifests,modules}
# mkdir -p /etc/puppet/environments/testing/{manifests,modules}
# mkdir -p /etc/puppet/environments/development/{manifests,modules}
```

接着追加“*”到/etc/puppet/autosign.conf配置文件中，让Master可以自动授权Agent的访问。

```
echo "*" > /etc/puppet/autosign.conf
```

最后启动Master守护进程。由于我们使用了个性化的安装，Puppet提供的启动脚本并不适用这种安装方式，所以需要通过以下方式启动Master的守护进程。

```
# /usr/local/puppet/bin/puppet master --daemonize >> /tmp/puppet_master.log 2&>1  &
```

启动Puppet守护进程后，不要忘记再次通过netstat –tnl | grep 8140命令来确认Puppet的守护进程的端口是否存在。如果存在则表明已经成功的启动Puppet，如果不存在则表明

启动 Puppet 失败，可以通过启动命令追加 --versose 和 --debug 参数方式来定位启动失败原因。

18.2 Puppet 辅助工具

18.2.1 Puppet Forge

我们再来回顾一下 Puppet Forge。Puppet Forge 是一个免费的 modules 基础模块仓库，它已经集成在 Puppet 源码中，所以无须再次安装。Puppet Forge 可以通过 Web 或 puppet modules 命令来获取这些基础模块。我们不但可以从 Puppet Forge 获取所需要的基础模块，也可以将自己开发好的基础模块发布到网上，分享给他人使用。以下为 Puppet Forge 的 puppet modules 命令方式获取与创建基础模块的方法。

1）创建标准 Puppet 模块。当我们使用了 Puppet 一段时间后，如果想将自己的通用 Puppet 模块分享给别人，可以通过“puppet module generate + 模块名”的方式来创建标准的模块目录结构，并通过 Puppet Forge（https://forge.puppetlabs.com/）注册账号，上传基础模块分享给他人。以下为创建 Puppet 标准模块的方式。

```
# puppet module generate example-nginx
Puppet uses Semantic Versioning (semver.org) to version modules.
What version is this module? [0.1.0] (模块版本号)
-->
Who wrote this module? [example] (模块名称)
-->
What license does this module code fall under? [Apache 2.0] (是否遵循 Apache2.0 协议)
-->
How would you describe this module in a single sentence? (模块描述)
-->
Where is this module's source code repository? (模块的源码库)
-->
Where can others go to learn more about this module? (获取更多的帮助信息)
-->
Where can others go to file issues about this module? (问题的反馈渠道)
-->
----------------------------------------
{
  "name": "example-nginx",
  "version": "0.1.0",
  "author": "example",
  "summary": null,
  "license": "Apache 2.0",
  "source": "",
  "project_page": null,
```

```
  "issues_url": null,
  "dependencies": [
    {
      "version_range": ">= 1.0.0",
      "name": "puppetlabs-stdlib"
    }
  ]
}
---------------------------------------
```

. 2）查找基础模块。通过“puppet module search + 基础模块名”的方式来查找基础模块。查找到的基础模块可以通过“ puppet module install + puppetlabs- 基础模块名”的方式进行安装。

```
# 查找基础模块
# puppet module search apache
# 安装基础模块
# puppet module search puppetlabs-apache
```

3）升级基础模块。

```
# puppet module upgrade puppetlabs-apache
```

4）删除基础模块。

```
# puppet module  uninstall puppetlabs-apache
```

当我们掌握 Puppet 后，通常可以使用 Puppet Forge 来查询与安装基础模块，这些基础模块在安装与管理系统软件包时已经充分考虑了性能与各发行版本的兼容性等，为我们节约了很多宝贵的时间，同时提升了工作的效率。

18.2.2 Example42

与 Puppet Forge 一样，Example42 也是一个免费的基础模块仓库，但与 Puppet Forge 不同的是 Example42 可以一次性将所有的基础模块克隆到本地，方便我们使用。

1. Example42 对操作系统发行版本的支持

Example42 是 Puppet 的基础模块仓库，以下是 Example42 支持的系统发行版本。当前 Example42 基础模块支持常见的操作系统发行版本。

- ❑ RedHat/Centos 5 和 6
- ❑ Scientific Linux 6
- ❑ Debian6 和 7
- ❑ Ubuntu 10.04 和 12.04
- ❑ OpenSuse 11 和 12（部分基础模块支持）

- Suse Enterprise Linux 11（部分基础模块支持）
- Solaris 11（部分基础模块支持）

另外，很多基础模块扩展支持以下系统发行版本：

- Amazon Linux 3
- Fedora
- Mint

2. Example42 自身版本

目前 Example42 分为三阶段版本，它们分别如下：

- OLD 模块（版本 1.x），此版本支持 Puppet 2.6 之前的版本，由于 Puppet 2.6 之前的版本比较老，所以这里笔者并不推荐大家使用。
- NextGen 模块（版本 2.x），此版本支持 Puppet 2.6 之后的版本，我们可以在 Git 的 submodules 中找到它，推荐使用。
- StdMod 模块（版本 3.x）example42 模块下的演化，坚持 stdmod 命名标准。此版本支持 Puppet 2.7 版本或更高的 Puppet 3.x 版本，推荐使用。

3. Example42 安装

Example42 的安装非常简单。目前 Example42 已经将代码托管到了 Github[⊖] 上，我们可以通过 Git 命令将 Example42 的代码克隆到本地。克隆 Example42 两个版本的方式如下：

```
# 方式 1：克隆 OLD 模块
# git clone --recursive -b 1.0 git://github.com/example42/puppet-modules.git
# 方式 2：克隆 NextGem 和 StdMod 模块（推荐）
# git clone --recursive git://github.com/example42/puppet-modules.git
```

笔者使用方式 2，将所有的基础模块克隆到本地的 puppet-modules 目录中如图 18-1 所示。

```
[root@localhost puppet-modules]# ls
00_example42scripts  common      firewall  lighttpd  mysql      oracle      puppet      samba      ssmtp      users
activemq             concat      foo       link      nagios     pam         puppi       sarg       stdlib     vagrant
apache               controltier foreman   logrotate network    php         rails       selinux    synbak     varnish
apt                  cron        git       lsb       nfs        phpsyslogng README      sendmail   sysctl     virtualbox
autofs               dashboard   hardening mailscanner nrpe     portmap     repo        snmpd      sysklogd   vmware
backup               dhcpd       hosts     mailx     ntp        postfix     resolver    spamassassin syslog-ng vsftpd
bind                 dovecot     iptables  mcollective openldap postgresql rootmail   sqlgrey    tftp       wordpress
clamav               drupal      jboss     monit     openntpd   powerdns    rpmbuild    squid      timezone   xinetd
cobbler              example42   jenkins   monitor   openssh    psad        rsync       squirrelmail tomcat    yum
collectd             exim        LICENSE   munin     openvpn    psick       rsyslog     ssh        trac       zip
```

图 18-1 Example42 模块目录

⊖ Github 官方网站 http://github.com，它是开源代码库的管理系统，Github 目前拥有 140 多万开发用户。随着越来越多的应用程序被转移到了云上，Github 已经成为管理软件开发以及发现已有代码的首选方法。

注意 如果 Git 命令找不到，需要通过 yum install git 方式来进行安装 Git 的环境。

除了 Github，Example42 已加入了 Puppet 的 Forge 项目，我们也可以通过 Puppet modules 命令找到 Example42 的相关模块并进行安装。

```
# 查找 example42 的模块
# puppet module search example42
# 安装 example42-nginx 模块
# puppet module install example42-nginx
```

18.3 快速构建企业内部网

18.3.1 企业内部网介绍

某公司主要从事手机网络视频和视频广告等业务，目前公司人数 200 人左右，由一名开发工程师兼网络管理员来负责系统的维护工作。由于移动互联网的快速发展，公司老板看到了商机，并认识到了现有公司网络架构的缺陷与不足，所以老板提出要求，需要对现有系统架构进行升级以满足后续企业的快速发展。兼职的网络管理员设计出了一个网络架构并最终得到了老板的认可。从成本角度出发，在不招聘新人的情况下，通过 Svn+Puppet 架构完成了企业内部网的构建，如图 18-2 所示。

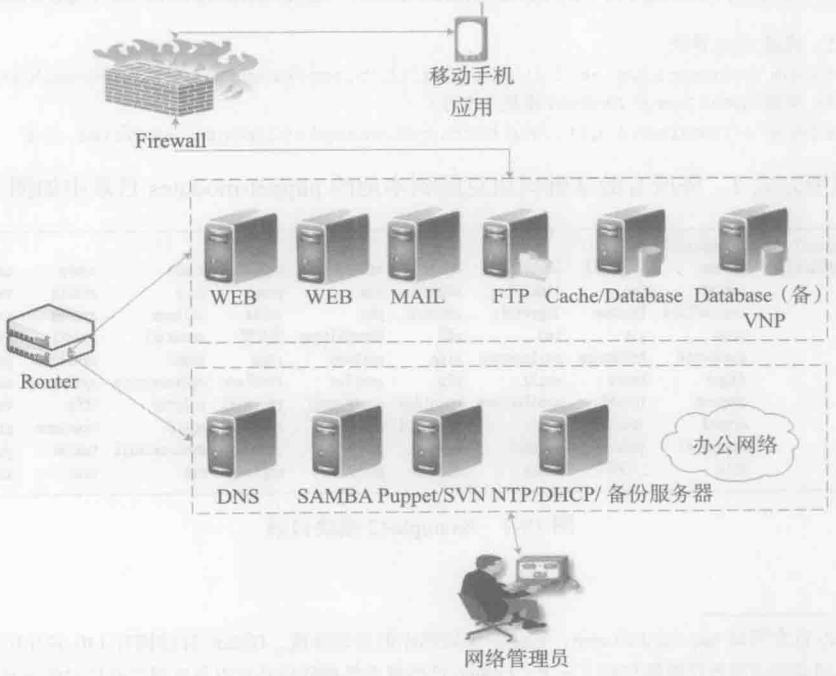


图 18-2 企业内部网

从以上的架构图我们可以了解到访问流程。公司的产品主要是移动网络视频与广告，移动端流量通过公司网络防火墙进入公司内部网，内部网分两个网段，两个网段分别通过路由器连接。网段 1 为公司自建 IDC 网络，存放了公司的所有服务器。网段 2 为办公网络，包含公司内部员工与一些提供基础设施的服务器。为了降低服务器的成本，网络管理员还将部分服务器作了混合部署。企业内部网结构如表 18-1 所示。

表 18-1 企业内部网结构

服 务	Hostname	IP
自建 IDC 网络		
防火墙	firewall.idc.example.com	192.168.1.2
web	web.idc.example.com	192.168.1.7
web/mail	mail.idc.example.com	192.168.1.8
cache	cache1.idc.example.com	192.168.1.17
cache	cache2.idc.example.com	192.168.1.18
db	db1.idc.example.com	192.168.1.37
db	db2.idc.example.com	192.168.1.38
ftp	ftp.idc.example.com	192.168.1.42
办公网络		
dns	dns.work.example.com	192.168.2.2
samba	samba.work.example.com	192.168.2.3
puppet/svn	puppet.work.example.com	192.168.2.4
ntp/dhcp/ftp 备份服务器 / 包管理	other.work.example.com	192.168.2.5
办公网络	work.example.com	192.168.2.50 – 192.168.2.250

在本书第 17 章 Puppet 管理 SSO 实践中笔者曾介绍了 Hostname 的用途。Hostname 在 Puppet 海量服务器管理中起到了很大的作用。但是为什么本节只使用了两个 Hostname（*.work.example.com 和 *.idc.example.com）？这里主要按照网络功能来划分 Hostname，当然也可以按照第 17 章的方式将 Hostname 分得更细。但是从表 18-1 中了解到每个服务器的功能比较单一，而且部分服务器为了节约成本对服务器的混用进行了支持。假如通过 Hostname 来划分，并不能区分混用的情况，而且划分得更细，会提高网络管理员的工作量，所以在这种服务器比较少且功能比较单一的情况下，通过网络功能划分 Hostname，并在后续的 Puppet 代码中，通过 IP 匹配的形式来管理整个企业内部网，工作成本会更低。

18.3.2 构建企业内部网

目前网络管理员已经搭建好了 Puppet，并安装了 Example42 的基础模块。本节笔者介绍企业内部网的 Manifests 的逻辑配置与个性化配置。通常在配置系统前我们都需要创建系统的业务账号，通过业务账号启动系统的服务，提高系统安全。另外还要设置所有服务器的 Host，并指定 Host 到 Master 的 IP，最后才是安装系统的服务。而这些配置逻辑都可

以通过 Puppet 的 Manifests 来完成。以下为企业内部网的 Manifests（/etc/puppet/manifests/site.pp）配置逻辑的 3 个部分：

1）创建基础节点（base）。将网段 1（*.work.example.com）与网段 2（*.idc.example.com）公共配置部分，如创建业务账号、组和指定 Host 到 Master，抽象到此节点。

2）创建 idc.example.com（网段 1）和 work.example.com（网段 2），根据服务器的角色安装软件。

3）创建默认节点（default），设置默认策略。

```
# Puppet Manifests
# 第一部分
node base{
  # 创建 user_00 组
  group{'user_00':
  ensure => present,
  gid => '507',
}
# 创建 user_00 业务账号
user{'user_00':
  ensure => present,
  uid => '507',
  gid => '507',
}
# 创建 user_00 业务账号的宿主目录
file {'/home/user_00':
  ensure => directory,
  group => '800',
  owner => '800',
  require => User['test_00'],
 }
# 指定所有 Agent 的 Host 为 Master 所在的服务器 IP
host { 'puppet.work.example.com':
  ip => '192.168.2.4',
  ensure => present,
 }
}
# 第二部分
node /*/idc.example.com inherits base{
   # 安装防火墙
  if ($ipaddress == "192.168.1.2" ){
    include iptables
  }
  # Web 服务器，安装 PHP 与 Apache
  if ($ipaddress == "192.168.1.7" ) or ($ipaddress == "192.168.1.8" ){
    include php
    include apache
}
  # 邮件服务器 ，安装 Posifix(postfix 是 Wietse Venema 在 IBM 的 GPL 协议之下开发的 MTA（邮件传输代理）软件）
```

```
    if ($ipaddress == "192.168.1.8" ){
      include posifix
    }
    # 安装数据库
    if ($ipaddress == "192.168.1.37" ) or ($ipaddress == "192.168.1.38" ){
      include mysql
    }
    # Ftp 服务器，安装 vsfptd(vsftpd 是 very secure FTP daemon 的缩写，安全性是它的一个最大的特点)
    if ($ipaddress == "192.168.1.8" ){
      include vsftpd
     }
    }
    # 第三部分
    node /*/work.example.com inherits base{
      # Dns 服务器，安装 BIND（BIND（Berkeley Internet Name Domain）是现今互联网上最常使用的 DNS 服务器软件，使用 BIND 作为服务器软件的 DNS 服务器约占所有 DNS 服务器的 90%）
    if ($ipaddress == "192.168.2.2" ) {
      include bind
    }
    # Puppet 服务器与 Svn 版本控制服务器，安装 Puppet 与 svn
    if ($ipaddress == "192.168.2.4" ) {
      include puppet
      include svn
     }
    # 混用服务器：
    # 1)ntp(Network Time Protocol（NTP）是用来使计算机时间同步化的一种协议，它可以使计算机对其服务器或时钟源（如石英钟、GPS 等）做同步化，它可以提供高精准度的时间校正（LAN 上与标准间差小于 1 毫秒，WAN 上几十毫秒），且可通过加密确认的方式来防止恶毒的协议攻击）
    # 2)dhcpd (DHCP(Dynamic Host Configuration Protocol，动态主机配置协议）是一个局域网的网络协议，使用 UDP 协议工作，主要的用途是给内部网络或网络服务供应商自动分配 IP 地址，dhcpd 是一款实现 DHCP 协议的开源软件）
    if ($ipaddress == "192.168.2.5" ) {
      include ntp
      include vsftpd
      include dhcpd
     }
    }
    # 第四部分
    node default{
      notify {"this is default node":}
    }
    }
    # end
```

以上为 Manifests 的逻辑配置，针对某个服务的个性化配置，如为 Apache 增加 vhost 和为 PHP 增加 pear 模块等，这些配置 Example42 也已经为我们考虑到，只需增加辅助参数即可。如在 IP 192.168.1.7 上追加 vhost 和 pear 这两个配置操作如下：

```
    if ($ipaddress == "192.168.1.7" ) {
```

```
    # 安装 PHP
    include php
    # 安装 pear 模块
    include php::pear
  # 安装 Apache
    include apache
   # 追加增加 vhos.example.com 虚拟主机配置
     apache::virtualhost      { "vhost.example.com": templatefile => "virtualhost.
conf.erb" }

  }
```

Master 配置好后就可以在 Agent 来测试整个配置了。具体方法如下：

```
# puppet agent --server puppet.work.example.com --test
```

对于互联网产品或大型软件系统来说，大量的配置文件、各类软件包资源、资源版本属性、资源间的相互依赖关系等等，使得这些产品和系统随着规模的增长和时间的延续，变得越来越复杂和难以维护，困扰着大量的开发和运维人员。Puppet的出现则为广大运维和开发人员带来了一个系统化的解决方案，大大降低了在配置和资源管理上的复杂度和投入，技术人员不再需要关心太多配置和资源管理的细节，可专心投入到业务自身的研发上。

本书作者有长时间在大型互联网产品上使用Puppet的经验，本书结合了这些经验，系统而全面地介绍了Puppet强大的功能和原理，帮助读者快速掌握其使用方法和工作机制。本书亦可作为平时使用Puppet的一本专业参考书。

赵建春 腾讯社交网络运营部助理总经理

如何高效地实现大规模服务器集群的运维管理，如何设计一套高可用的配置管理系统，是摆在每个互联网运维工程师面前的难题。Puppet正是为了解决大规模集群配置管理而诞生的一款优秀的系统。作者凭借多年在腾讯海量运维积累下的Puppet最佳实践经验撰写的本书将Puppet从入门学习、灵活实践，到高阶应用的过程巧妙地融入各个篇章中，内容多为经验总结概括，实战性很强，值得每位运维工程师、运营开发工程师、系统架构师和广大Puppet爱好者捧读。

梁定安 腾讯社交平台业务运维负责人

上架指导：计算机/Linux
ISBN 978-7-111-48598-8
9 787111 485988 >
定价：69.00元

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn