TURING 图灵程序设计丛书

Apress®

Pro Puppet

精通Puppet配置管理工具

[澳] James Turnbull
[美] Jeffrey McCune 著

高永超 译

- Puppet Labs专家力作
- 配置管理实战指南
- 云计算时代系统管理员必备

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Pro Puppet

精通Puppet配置管理工具

[澳] James Turnbull
[美] Jeffrey McCune 著
高永超 译

人民邮电出版社
北京

图书在版编目（CIP）数据

精通Puppet配置管理工具 /（澳）特恩布尔(Turnbull,J.)，（美）麦丘恩（McCune,J.）著；高永超译. -- 北京：人民邮电出版社，2012.5
（图灵程序设计丛书）
书名原文：Pro Puppet
ISBN 978-7-115-27951-4

Ⅰ. ①精… Ⅱ. ①特… ②麦… ③高… Ⅲ. ①程序开发工具 Ⅳ. ①TP311.52

中国版本图书馆CIP数据核字(2012)第070120号

内 容 提 要

本书系统介绍了开源配置管理工具Puppet，并提供了帮助使用Puppet的大量资源。书中讲述了如何创建Puppet recipe、扩展Puppet并使用Facter整合来自服务器的配置数据，同时讲述了如何使用Puppet管理Postfix、Apache和MySQL服务器，以及如何加载平衡Puppet Master。

书中涵盖了安装、使用并利用Puppet进行开发所需要的全部知识、内部技巧和技术。非常适合系统管理员、操作人员和开发人员阅读。

图灵程序设计丛书

精通Puppet配置管理工具

◆ 著 [澳] James Turnbull [美] Jeffrey McCune
译 高永超
责任编辑 朱 巍
执行编辑 李 瑛

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：17.25
字数：408千字 2012年5月第1版
印数：1-3 500册 2012年5月北京第1次印刷
著作权合同登记号 图字：01-2011-7476号
ISBN 978-7-115-27951-4

定价：69.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

献　　词

献给一直是那么优秀的我的搭档和好朋友，Ruth Brown。

James Turnbull

献给我的父母，Pete和Gloria，他们年复一年地容忍了我在陪他们度假时依然沉浸在笔记本电脑中。同时还要献给Dave Alden，作为一位良师益友，他教给了我关于配置管理方面的知识，并鼓励我更深入地学习。

Jeff McCune

作译者介绍

James Turnbull　开源拥趸，Linux Australia前任主席，经常在OSCON、Open Source Bridge、DevOpsDays等大会上发言。现任职于Puppet Labs。目前已有5本著作，均涉及开源软件。James是澳大利亚人，目前居住在美国奥勒冈州的波特兰。他的兴趣爱好十分广泛，包括烹饪、品酒、政治理论、新闻摄影以及哲学，最近还加入了波特兰Timbers协会足球队。

Jeffrey McCune　Puppet社区成员，开源软件的支持者，经常在苹果的世界开发者大会、Macworld、Open Source Bridge、Velocity等大会上发言。现任职于Puppet Labs，致力于编写代码和帮助客户改进他们的Puppet部署。Jeff目前居住在奥勒冈州的波特兰，喜爱钻研微控制器、动画、摄影、音乐，爱好徒步旅行以及沙滩长途步行。

高永超　专职SA，喜欢钻研架构和运维相关知识，有两年Puppet使用经验，现任豆瓣运维工程师。网络ID为flex，个人邮箱：flex@flib.me。

技术审稿

Jes Fraser　一位来自于新西兰的专门从事企业Linux和Puppet应用的顾问。她喜欢唱歌、弹奏钢琴，当然还有写作。

前　言

“循环调用SSH命令不是一个我能接受的解决方案。”

—— Luke Kanies，Puppet开发者

系统管理员和操作人员的生活往往都离不开解决一系列重复性任务：配置主机、创建用户以及管理应用程序、守护进程和服务。通常在一台主机的生命周期内，这些任务会被重复很多次，从搭建系统到服务器下架，以及为了修正错误或进行整理而对配置进行添加或修改。

这些重复性任务的通常解决方案是试图使用脚本和工具进行自动化操作，这就需要开发定制的脚本和程序。刚成为系统管理员时，我编写了一系列控制语言（CL）和Rexx脚本来管理和操作不同类型的基础设施。这些脚本不仅复杂而且基本没有文档，完全是根据我自己的工作环境定制的。

我的经历比较有代表性，为了让工作更轻松，以便把更多的时间放在更有趣的项目和任务上（或者能早点去酒吧），自动化处理一些无聊和手工的任务是大家常用的应对之法。

在这些专门开发的脚本中，只有很少一部分会被发布、文档化或者重用。事实上，大部分定制脚本的版权取决于操作人员或者系统管理员所在的机构，并且随着工作的进行，这些脚本通常会被丢弃。这就导致了同样的工具被一次又一次地开发。甚至即使在同一个公司，如果前任的工作不合继任者的心意（有时候是因为太晦涩难懂），也会发生同样的情形。

这些定制的脚本和程序很少为了适应大型环境而进行扩展，并且在稳定性、适应性和功能上往往都存在很多问题。在多平台的环境下，这些脚本通常只能应用在一种目标系统上。这就导致了这样的情形：当需要一个创建用户的脚本时，BSD、Linux和Solaris都各自需要维护一个版本。你不得不消耗更多的时间和精力来编写和维护这些本是用来减少管理工作的工具。

当然还有其他一些途径，比如购买HP的Opsware、BMC的CONTROL-M、IBM的Tivoli套装工具或者CA的Unicenter等操作和配置管理工具。但是商业工具通常都有两个致命的问题：价格和适用性。尤其是价格方面，管理的平台和主机越多，价格就越高。在一个大的生产环境中，这些工具的许可费用可能达到数百万美元。

适用性也是一个主要问题。商业工具通常是不开源的，并且仅能使用它们提供的特性。这就意味着，如果你想扩展它们来对你的环境做一些定制什么的，就必须向别人申请为你开发一个新的特性。这可能会产生一段等待期，并带来相应的开销。考虑到组织中现存的大量各式各样的部署机制、平台、配置和应用，要找到一个可以完全适应自己环境的商业定制工具是非常困难的。

在内部开发和商业产品之外还有另一种选择：自由开源软件（FOSS）。自由并且开源的配置管理工具能带来两大好处：

- 它们开放并且可扩展；
- 它们是免费的！

使用FOSS产品，工具的源码就在你手上，你可以通过二次开发来增强或者调整它们。不用等待厂商来实现你需要的功能，也不需要为新特性和升级付费。作为用户和开发社区的一员，你可以分享对这个工具未来的想法。你和你的组织可以进而为这个想法作出自己的贡献。总之，你可以改变自己所用工具的发展方向，使其能够更灵活地适应你所在组织的需要。

就如刚刚提到的，获取任何一个工具时，需要着重考虑价格因素。有了自由开源软件，这就不再是问题。不用付出任何费用你就能获得软件以及它的源码。

当然，天下没有免费的午餐，这其中有什么奥妙呢？原来，不像商业软件，开源软件不能提供任何有保证的支持。但这并不意味着没有可用的支持：许多开源工具都拥有庞大和活跃的社区，社区成员通过邮件列表、论坛、Wiki和IRC等形式回答问题和提供帮助。

注意 包括Puppet在内的许多开源工具，同样有提供商业版本或者支持的机构。比如，本书的作者James Turnbull和合作者Jeff McCune都在Puppet Lab工作，该组织支持着Puppet的开发工作。

Puppet（http://www.puppetlabs.com/puppet）就是为了填补系统管理员、操作人员和开发人员使用的工具的缺口而诞生的。通过对基础设施进行更简单、容易和廉价的管理，来让这些人员的工作更加轻松。这本书将在介绍开源配置管理工具Puppet的同时，帮你熟悉Puppet的安装、配置以及如何将Puppet整合进你的环境。

致谢

我们需要感谢下面这些积极投入这个项目并提供宝贵见解的人们：

Dan Bode

Luke Kanies

Nigel Kersten

Dennis Matotek

Hal Newton

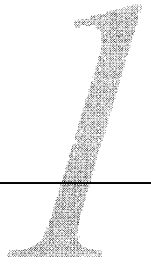
R.I. Pienaar

Trevor Vaughan

以及所有努力让Puppet变得更酷的Puppet Labs团队。

目　录

第1章 开始使用Puppet



Puppet是一个用来管理计算机系统配置的开源框架和工具集合。本书将介绍如何使用Puppet来进行配置管理。在随后的内容中，我们将介绍Puppet的相关特性以及展示如何将Puppet整合到配置和管理周期中去。为此，在第2章我们将带你配置一个真实的解决方案。

在这一章，我们首先快速浏览一下Puppet的概况，包括什么是Puppet，它如何工作以及应该使用哪个版本等，然后向你展示如何安装Puppet和它使用的盘点工具：Facter。我们将说明如何在Red Hat、Debian、Ubuntu、Solaris、Microsoft Windows等操作系统中安装Puppet，以及如何使用Ruby Gem来安装Puppet。然后我们将对它进行配置，并向你展示如何创建第一个配置项。与此同时，我们还会介绍“模块”这一概念，它是Puppet用来收集和管理不同类型配置数据的方式。最后我们将展示如何使用Puppet agent将这些模块应用到一台主机上。

1.1 什么是 Puppet

Puppet是一个基于Ruby，并使用GPLv2协议授权的开源软件，它既能以客户端-服务端的方式运行，也能独立运行。它主要由Luke Kanies和他的公司Puppet Labs（以前称为Reductive Labs）开发。Kanies从1997年开始涉足Unix和系统管理，然后基于这些经验开发了Puppet。因为对已经存在的配置管理工具不满意，Kanies在2001年开始了这一新工具的开发，并于2005年创立了一家专注于自动化工具的开源软件开发公司：Puppet Labs。不久之后，Puppet Labs发布了他们的旗舰产品Puppet。

Puppet可以用来管理UNIX（包括OSX）和Linux平台，并且最近又添加了针对Microsoft Windows的支持。Puppet通常可以用来管理一台主机的整个生命周期：从初始化到安装、升级、维护以及最后将服务迁移并下架。Puppet被设计为能够持续与主机进行交互，而不是仅仅提供一个只负责搭建主机却并不管理它们的工具。

Puppet拥有一个简单并且容易理解和实施的操作模型（见图1-1）。这个模型由三部分组成。

- 部署
- 配置语言和资源抽象层
- 事务层

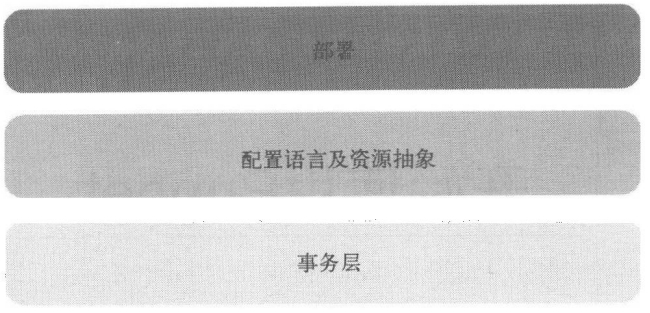


图1-1　Puppet模型

1.1.1　部署

Puppet通常使用简单的客户端-服务端模型(图1-2)进行部署。服务端被称为“Puppet master”，客户端软件被称为agent，主机本身则被定义为一个节点。

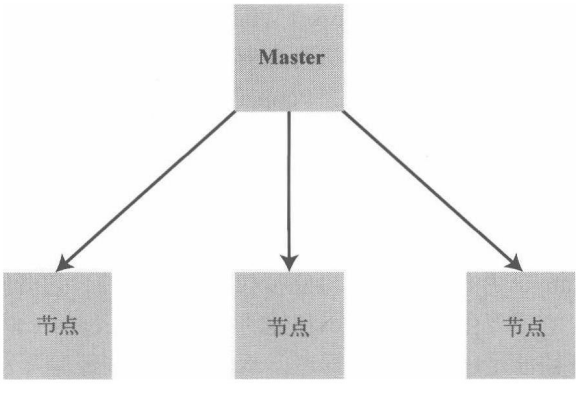


图1-2　Puppet客户端-服务端模型

Puppet master在一台主机上以守护进程的方式运行，它包含了环境所需的所有配置。Puppet agent则通过一个使用标准SSL协议进行加密和验证的连接与Puppet master进行通信，然后接收或者“拉取”需要被应用的配置。

很重要的一点是，Puppet agent在已经获得了需要的配置或者没有任何可以被应用的配置时不会做任何事情。这意味着Puppet只会在需要时对你的环境作出变更。这整个过程被称为一次配置运行。

每一个客户端既可以通过守护进程的方式来运行Puppet (比如使用cron)，也可以手动启动。通常的做法是以守护进程的方式运行Puppet，并周期性地与master进行通信，以此来保证配置已经更新到最新并且能及时接收新的配置。不过，也有很多人觉得使用cron或者手动运行Puppet更符合他们的需求。在默认情况下，Puppet agent会每30分钟与master进行一次通信，检查新添加的或者已改变的配置。你可以自行设定这个周期来适应你的环境。

当然也存在其他部署模型。比如，Puppet也能抛开Puppet master以独立方式运行。在这种模式下，配置放置在被管理的主机上，然后通过手动运行puppet程序来执行和应用这些配置。我们将在稍后讨论这种模式。

1.1.2 配置语言和资源抽象层

Puppet使用一种描述性语言来定义配置项，配置项在Puppet中被称为“资源”。这种描述的本质使得Puppet和许多其他配置工具之间产生了重要的差别。描述性语言可以声明你的配置的状态——比如，声明一个软件包应该被安装或者一个服务应该被启动。

大部分配置工具，如shell或者Perl脚本，是命令式或者过程式的。它们描述的是事情应该怎么做而不是所需要的最终状态应该如何——比如，大部分用来管理配置的定制脚本都应当被看作命令式的。

这就意味着Puppet的用户只需要声明他们的主机应该处于什么状态即可：比如什么软件包应该被安装，什么服务应该被运行。使用Puppet，系统管理员不需要关心如何达到这种状态——那由Puppet负责。相反，我们关心的是如何将主机的配置抽象到一个个资源中。

1. 配置语言

描述性语言的实质是什么？让我们看一个简单的例子。这里有一个包括Red Hat企业版Linux、Ubuntu以及Solaris的主机环境，我们需要在所有的主机上安装vim程序。如果手工来做，我们需要写一个脚本来完成下面这些事情：

- 连接到目标主机（包含输入密码或者提供密钥）；
- 检查是否安装了vim；
- 如果没有，使用每个平台上适当的命令来安装它，比如在Red Hat上使用yum命令，在Ubuntu上使用apt-get命令；
- 可能还包括报告结果以确保命令成功完成。

注意 如果你需要升级已经安装的vim或者应用vim的一个特定版本的时候，上面的步骤会变得更加复杂。

Puppet使用不同的方式来完成这一过程。我们在Puppet中为vim包定义一个配置资源。每个资源都由一个类型（表明被管理的是什么样的资源：软件包、服务或者定时任务等）、一个标题（资源的名字）以及一系列属性（用来说明资源状态的值——比如服务是被启动还是被停止）组成。

代码清单1-1是一个资源的例子。

代码清单1-1 一个Puppet资源

```
package { "vim":
    ensure => present,
}
```

代码清单1-1中的资源表明了vim包应该被安装。它的结构是这样的：

```
类型 { 标题:
    属性 => 值,
}
```

在代码清单1-1中，资源的类型是package。Puppet默认提供许多资源类型，可以用来管理文件、服务、软件包以及定时任务等。

注意　可以在http://docs.puppetlabs.com/references/stable/type.html找到Puppet目前能管理的全部资源类型及其属性。我们将在第10章讨论如何扩展Puppet来支持额外的资源类型。

接下来是资源的标题，在这里就是我们要安装的软件包的名字：vim。资源的类型和标题组合在一起构成一个针对这个资源的引用。例如，刚刚的资源可以被称为Package["vim"]。我们在接下来的章节介绍如何创建资源之间的关联时将看到许多类似的引用。这些关联可以将我们的配置结构化，比如在启动服务前安装相关联的软件包。

最后，我们指定了一个单独的属性，ensure，它的值是present。Puppet通过属性得知我们对配置资源所要求的状态。每一个资源类型都有一系列可配置的属性。这里ensure属性用来指明软件包的状态：已安装、已卸载等。而它的值是present则告诉Puppet我们希望安装这个软件包。如果要卸载这个软件包,我们只需把它的值变为absent。

2. 资源抽象层

资源被创建后，当agent连接到master时，Puppet将负责在管理这些资源时产生的一些细节问题。Puppet知道不同平台和操作系统在管理特定资源类型时的差异，并据此来处理这些细节问题。每一种资源类型都有许多“提供者”。一个软件包资源的提供者包含了“如何”使用特定的软件包管理工具来管理软件包。比如，对于软件包这一资源类型来说，有超过20个的提供者，覆盖了包括yum、aptitude、pkgadd、ports和emerge在内的多种包管理工具。

当一个agent连接到master时，Puppet使用一个叫做“Facter”的工具来返回agent的相关信息，其中就包括了主机运行的操作系统。然后Puppet会根据这一点为这个操作系统选择合适的软件包提供者，并使用这个提供者来检查vim包是否已经安装。例如，在Red Hat上它会执行yum命令，在Ubuntu上会执行aptitude命令，而在Solaris上会使用pkg命令。如果没有安装这个软件包，Puppet就会安装它。反之，则什么都不做。

最后Puppet会向Puppet master报告应用这些资源配置是否成功。

FACTER和FACTS

Facter是一个系统盘点工具，本书会经常用到它。它返回每个agent的“fact”，比如agent的主机名、IP地址、操作系统和版本以及其他配置项。这些fact由agent在运行的时候进行收集，然后发送给Puppet master，并自动被创建为可以被Puppet使用的变量。

可以通过在命令行下运行facter程序来查看客户端上所有可用的fact。每一个fact都返回一

个键值对。例如：

```
operatingsystem => Ubuntu
ipaddress => 10.0.0.10
```

这样我们就能使用这些值来单独配置每一台主机。比如，在获得一台主机的IP地址后，我们就能据此配置这台主机的网络设置。

这些fact可以通过变量的形式在Puppet的配置中使用。结合预先定义好的Puppet配置和这些变量，就可以为每一台主机定制配置。例如，你可以编写一些通用资源，比如网络设置，然后使用客户端返回的数据来定制它们。

Facter还能帮助Puppet理解如何在一个agent上管理特定的资源类型。比如，如果Facter告诉Puppet一台主机正在运行Ubuntu，Puppet就知道在这个agent上要使用aptitude命令来安装软件包。Facter同样能进行扩展，可以通过添加自定义的fact来返回一些关于主机的指定信息。我们将在安装完Puppet之后安装Facter，并在后面的章节详细讨论Facter的一些细节。

1.1.3 事务层

Puppet的事务层就是它的引擎。Puppet事务涉及配置每一台主机的过程，包括：

- 解释和编译配置；
- 将编译好的配置同步到agent；
- 在agent上应用配置；
- 向master报告程序运行的结果。

Puppet运行的第一步是分析你的配置并且计算如何在agent上应用它们。为此目的，Puppet会创建一张图来表示所有的资源以及它们之间的关系，还有它们和agent之间的关系。Puppet将按照这些关系来决定每一台主机应用资源的顺序。这个模型是Puppet的强大特性之一。

接着Puppet为每一个agent取得相应的资源并将它们编译成为“目录”。然后将目录发送到各个主机并通过Puppet agent来应用它们。最后程序运行的结果以报告的形式发回给master。

事务层允许配置在主机上被重复创建和应用。这称为幂等，意思是多个程序的相同操作会导致同样的结果。Puppet配置可以在主机上安全地运行多次，并且每次运行都能得到同样的结果，Puppet使用这一点来保证配置的一致性。

不过Puppet并不具有完全的事务性。因为事物并没有被记录（除了日志），所以无法像一些数据库那样对事务进行回滚。不过你可以使用无操作模式（noop）的事务模型来测试配置的执行，这不会导致实际上的改变。

1.2 选择正确的 Puppet 版本

Puppet的最佳版本通常都是最新的版本，在撰写本书时最新的发布版是2.6.x，更新的版本

也正在开发中。在2.6.x版本中最大的改进是使用REST API替换了XML-RPC作为传输层，这极大地提高了性能。并且在运行稳定表现良好的同时，2.6.x还包含了许多之前版本所没有的新特性和功能。

为什么Puppet改变了版本号的数字记法

如果你熟悉Puppet的开发，就会发现Puppet的发布版本号从0.25.5直接跳到了2.6.0。这中间发生了什么？——是2.6.0版比0.25.5版强大和稳定11倍吗？答案可以说是也可以说不是。2.6.0版包含了大量附加的特性并且移除了最后的XML-RPC传输层。重要的是，发布版本号的跳跃表明之前的版本号并不能准确反映Puppet的发展和变化。就稳定性和功能性而言，0.24.x版本和0.25.x版本的小数位都应该向右移。另外，在0.25.0版本之后，Puppet就不再像之前的版本号表明的那样，是一个“前1.0版”的产品。

Puppet早期发布的版本，尤其是0.24.x版之前，只有非常少的特性，并且还有大量的缺陷和问题。它们在很大程度上已经不再被支持，对于0.20.x，0.22.x和0.23.x或更早版本的帮助请求，多半的建议都是升级它们。我们不推荐你使用这些版本。

注意　即使绝大部分材料（除了我们特别指明的）都能向前兼容到0.24.7，本书依然假设你使用2.6.x或者更新的版本。要记住，如果你使用0.24.7或者0.24.8版，你将无法获得0.25.x和以后版本在性能提升方面所带来的好处。

在各种操作系统上，都有许多很旧的Puppet版本。其中0.24.x版是最广泛的，2.6.x和0.25.x版只在较新的操作系统和平台上被打包和发布了。如果不能在你的发行版上找到较新的Puppet版本，你可以自行制作软件包、backport版本或者从源码进行安装（虽然我们不推荐最后这种方法——原因见下文）。

1.3　我能混用 Puppet 的版本吗

最常见的Puppet部署模型是客户端-服务端模型。许多人询问是否能使用不同的Puppet版本作为master和agent。答案是可以，但前提是要遵照一些注意事项。第一点要注意的是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，但是反过来不行。

第二点要注意的是，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基本不可能正确工作。通常，0.24.x版的agent都能正常连接到2.6.x和0.25.x版本的master并且工作正常。更新版本的master就可能无法完全兼容早期的agent，一些功能和特性可能会出现异常。