TURING 图灵程序设计丛书

Pro Puppet Second Edition

精通Puppet配置管理工具

（第2版）

【美】Spencer Krum
【美】William Van Hevelingen
【美】Ben Kero 著
【澳】James Turnbull
【美】Jeffrey McCune
李超 译

人民邮电出版社
北京

致　　谢

这本书能够问世，我要感谢的人实在是太多了！感谢爸爸和妈妈容忍我长时间投入写作，过去这段时间，写书几乎占用了我所有的业余时间。感谢奶奶和爸爸让我从小就接触到计算机，感谢爷爷和奶奶让我圆了大学梦。感谢 Alderman 太太、Ashley、Beverley 和 Ryan 鼓励我写书。感谢 Corbin，在我还小的时候，他就在技术上和生活上影响着我，并一以贯之地帮助我。

感谢 Janaka 创建了 TheCAT。感谢 Marut、Finch、CATastrophe、Johnj、Hunner、Jesusaurus 以及 TheCAT 的每个人，是你们教给了我所有的一切。感谢 Plaid Pantry 便利店凌晨 4 点还在热情服务的 Donkey，还有，是他教会了我有关 IP 的知识。感谢 Greg Haynes，从我们认识的第一天起他就不断地教导我，把我领进了技术世界。感谢 Epitrope 让我开始写作这本书。

感谢 Gary Kilmocwiz、Mike Kinney、Scot Lambert、Charles Hill、Lance Coombs 以及 UTi DevOps 的其他同事，你们在我写书期间对我十分忍让，你们是世界上最好的同事。

感谢本书的合著者对我的帮助，感谢本书的技术评审，没有你们就不会有这本书。

特别感谢 Krinkle（Colleen Murphy）对我糟糕的英语文法进行编辑润色。不论多忙，你总会抽出时间来仔细阅读我的书稿。特别感谢 Nightfly（Sage Imel）帮助我处理了书稿中的一些棘手的部分。没有二位的帮助，这本书的质量将大打折扣，对于你们为此所付出的大量时间和精力，在此深表谢意！

感谢 Puppet Labs 公司的同事们给了我巨大的帮助，他们是：Dawn、Lee、Reid、Adrien、Hunter、Ben Ford、Eric0、Thomas Halgren、Brenan、Haus、Stahnma、Nigel、Nick F、Nick L、Henrik Lindberg、Jeff McCune、Kara Sowles、Kent Bye、Charlotte、Aliza，当然还有 Luke Kaines。没有 Puppet Labs 可能就不会有 Puppet，没有你们付出的努力，这本书就不可能有现在这个样子。

最后要感谢 Apress 出版社的编辑们为本书所做的工作：Steve、Christine、Michelle 和 James，你们提供的帮助使我们受益匪浅。还要感谢 Kevin，你如此善解人意，我们一定努力报答。

—— Spencer Krum

深深地感谢 Puppet Labs 卓越的工程师们，你们创建了一个强大的项目以及周围的生态系统。感谢 Puppet 社区的成员们，你们创造的工具总能带给我们惊喜，没有你们，这一切都不会发生。我特别为本书的合著者感到骄傲，没有你们我不可能有机会和热情完成这本书。最后，我还要感谢 Apress 出版社，正是由于你们所提供的一流服务，才使本书得以最终面世。

—— Ben Kero

感谢母亲，她总是一如既往地支持我。感谢父亲，是他引领我迈进技术世界。感谢 John Harris 教我如何作一名系统管理员，感谢 Scott Andre 做我的导师，感谢 Janaka Jayawardena 和 Reid Vandewiele 教我使用 Unix，感谢 Spencer 激励我做出了以前自己想都不敢想的成绩。最后，感谢 TheCAT 的各位前辈和新人，正是你们，让我明白了永远不要放弃学习。

—— William Van Hevelingen

前　言

很荣幸向大家介绍本书的新版本。自 2011 年 5 月本书第 1 版面世以来，这本书就在 Puppet Labs 参加的各种会议和技术活动中广受欢迎，始终处于供不应求的状态。

自本书第 1 版面世以来，很多事情发生了变化。那时 Puppet 2.7 刚刚发布，接着我们的第一个商业版本 Puppet Enterprise 发布了。

2011 年 5 月，Puppet Forge 中只有 170 个模块，现在已经有 1700 个模块，多数都由广大的、快速成长的社区贡献而来。说到社区，全球有超过 30 个 Puppet 用户组，定期组织会议讨论 Puppet 的方方面面。而在本书第 1 版面世时，这些用户组还都不存在。

作为一家公司，Puppet Labs 在本书第 1 版面世时只有 33 名员工，而今天已经有 200 人，我们能够为 Puppet 用户提供更多的资源和服务，包括我们的帮助站点ask.puppetlabs.com。每个人都可以在上面提出自己的问题，或者回答别人的问题。

对我来说，一个显著的变化是 Puppet 用户。几年前，Puppet 用户是一群为小组织工作的、聪明而富有创造力的系统管理员，今天，Puppet 用户不仅有这些聪明有创意的系统管理员，还包括大企业中的开发者和 IT 经理们。由 Puppet 管理的超过 5 万个节点的主机群不再少见。今天，Puppet 用户广泛分布在工业界的各个领域，包括银行、金融、科研、技术、零售、电子商务、政府、电信和互联网等，在任何希望通过 IT 技术拓展业务竞争力的地方，你都可能看到 Puppet 的身影。

Puppet 的开发初衷是作为一款系统管理员的工具，帮助他们专注于业务目标，而非没完没了的技术细节。计算机应该做那些枯燥重复的工作，而人应该做那些困难的（也是有趣的）工作，例如构建和维护大规模的 IT 基础设施。我希望人们不再重复解决同样的问题，而解决问题的关键就在于代码重用。我知道用户真正关心的是他们使用的服务，而不是提供这些服务的主机和技术细节。

于是我为 Puppet 创建了一种适于描述主机配置的简单声明式语言，系统管理员可以用它简单快速地告诉任何节点（不论这个节点是有物理实体的服务器，还是虚拟机、交换机或路由器等）它们的任务是什么，要达到什么样的目标，而不必关心实现目标的具体步骤是什么。

过去我看到的是不断增多的多种形态的数据中心，我希望更进一步，把这些数据中心变成正在运行的软件。我们从管理服务器和工作站开始，但现在我们的管理目标还包括了交换机、路由器、防火墙、存储阵列等多种形态的 IT 设施。API 驱动的云世界正在缩短 IT 设施和用户间的反馈链，给技术生态圈带来了更大的时间压力，系统管理员和他们的运维团队必须正视这个挑战，为用户带来更好的技术和更可靠的反馈支持，最大程度地发挥 IT 技术的潜能。

Puppet Labs 正是引领这一潮流的组织之一，过去几年中 Puppet 的技术变革是我们努力成果的一部分，本书将为每一个希望充分利用 Puppet 潜能的读者提供巨大价值。

这一新版本包含了如下主题。

Hiera：Hiera 是一个存储和查询节点特有数据的强大工具，本书新增了一整章来专门介绍它。

Puppet 将集群和节点特有数据存入 Hiera，从而实现了数据和代码的分离，Puppet 可以通过它提供的可配置层次数据结构方便地配置节点。在 Puppet 模块中使用 Hiera 封装私有数据，使通过模块进行协作变得更容易。新版本全面介绍了 Hiera 的各个方面，包括它的高级合并特性。

Geppetto：Geppetto 是一个基于 Eclipse 的 Puppet 代码集成开发环境。本书根据 Geppetto 的最新版本对原有内容进行了更新，介绍了它与外部服务（如 Puppet Forge）的集成等特性。

MCollective：面对日益动态化、自服务化的 IT 环境，我们很难简单地通过静态配置达到管理基础设施的目标。与资源动态互动的能力显得越来越重要，MCollective 正是实现这一目标的利器，不论做简单的服务维护，还是复杂的应用批量部署，都可以通过 MCollective 提供的多种粒度的访问功能来实现。借助于它的增量部署方式，我们对整个过程的掌控能力得到了显著提升。本书根据 MCollective 2 对原有内容进行了更新。

Puppet 3：Puppet 3 于 2012 年发布，对原有版本进行了很多改进和提升，包括大幅缩短了 agent 的运行时间，更快的条目编译速度，与 Hiera 的无缝集成等。本书完全基于 Puppet 3 版本，所有示例代码都通过了 Puppet 3 环境的测试。

PuppetDB：在中央存储服务 PuppetDB 的帮助下，你的集群节点数量可以轻松翻番。PuppetDB 基于 PostgreSQL，拥有一套功能丰富的 API，并催生了一些类似于 Puppetboard 这样的附属开源项目。本书从安装、配置到使用，对 PuppetDB 进行了详细介绍。

企业版 Puppet 控制台和 Puppetboard：企业版 Puppet 控制台是一个图形用户界面工具，作为管理和操作 Puppet 基础设施的首选工具，它的功能包括报表、错误排查和分析、资源发掘与对比等。Puppetboard 是一款管理 PuppetDB 的 Web 应用，基本上替代了原来的 Puppet Dashboard，具备 Puppet Dashboard 的报表功能。这两个工具都是在本书第 1 版出版后出现的。另外本书还介绍了 Red Hat 的一个功能类似的项目：Foreman。

附属工具：Puppet 拥有大量附属工具，这些工具适用于各种操作系统和平台，使用了多种多样的技术。本书在第 1 版的基础上根据这些工具的最新进展进行了更新，并增加了一批新工具，包括 `puppet-lint`、`puppet-rspec`、`travis-ci` 集成工具、`r10k` 以及 `puppet-librarian` 等。更新后的章节详细介绍了 Puppet 模块工具，用户借助它可以在命令行中搜索 Puppet Forge 上的模块，打印列表、升级、安装模块、解析模块依赖以及安装依赖模块等。

横向扩展：Puppet 及其相关软件具备强大的扩展能力，本书通过一个完整的实例介绍了 Puppet 的传统扩展方法，以及独立模式下运行 Puppet 等较新的扩展方式。尽管企业版 Puppet 由于自身已经具备了很好的扩展能力，很少使用这些扩展方法，但它们在一些特定情境中非常有用。

如果这是你初次接触 Puppet，欢迎加入我们，希望你充分利用我们提供的免费资源，包括：

- Puppet 在线问答；
- Google Group 上的 Puppet 用户组；
- Puppet 文档中心；
- Puppet 训练中心；
- 我们的 IRC 频道，#puppet 中的 nibalize、blkperl 和 bkero 是本书作者，欢迎随时与他们联系。

如果你是 Puppet 的老用户，我相信这本升级版能够帮助你更深入地了解 Puppet。多用用上面的免费资源，尝试一下那些你还没用过的新工具。

愿你遇到的所有难题都能轻松搞定，享受这个充满挑战的旅程吧！

——Luke Kanies，Puppet Labs 公司创始人兼 CEO

目　录

第1章 Puppet初体验

Puppet是用来管理计算机系统配置的开源框架和工具集。本书将向你介绍如何使用Puppet进行配置管理和Puppet的特点，并通过从第2章引入的一个真实场景，来说明如何将Puppet整合进预安装和管理流程。第3章展示如何通过版本控制和Puppet环境来控制Puppet流程。第4章介绍实现Puppet基础设施的高可用性及横向拓展性的方法。本书的其他部分将主要关注扩展Puppet的用途和它的工具生态环境，以及如何实现高度直观地反映基础设施的状况。

本章将介绍如下内容：

- Puppet概述，它的概念、工作原理以及版本选择；
- 如何使用RubyGems在Red Hat、Debian、Ubuntu、Solaris、Microsoft Windows以及Mac OS X上安装Puppet及其附属工具Facter；
- 如何配置Puppet，编写配置项；
- 用于编写Puppet配置项的DSL（领域特定语言）；
- Puppet实现收集和管理配置信息的方法——模块；
- 如何使用Puppet agent运行一个模块。

1.1 什么是 Puppet

Puppet是一个基于Ruby语言的配置管理工具，使用Apache 2.0许可证分发。它有两种运行模式：服务器端-客户端模式和独立运行（stand-alone）模式。Puppet最初由Luke Kanies开发，现在由他的公司Puppet Labs开发。Kanies从1997年开始从事Unix系统管理工作，并基于此经验开发了Puppet。由于对已有的配置管理工具不满意，Kanies从2001年开始这方面的工具开发工作，并于2005年创立了Puppet Labs，一家专注于研究自动化工具的开源技术开发公司。不久，Puppet Labs发布了旗舰产品Puppet Enterprise（企业版）。Puppet有两个版本：开源版和企业版。企业版包含一个自动安装工具、一个Web管理界面以及技术支持。本书主要介绍开源版，但由于这两个版本使用了相同的内核，因此这些知识对两者都适用。

Puppet可以管理Unix（包括OS X）、Linux和Microsoft Windows平台上的配置，它可以管理一台主机的整个生命周期：安装、升级、维护以及报废。Puppet的特点在于能与主机持续交互，而不像有些预安装工具那样，安装完系统后就不管了。

Puppet有一个易于理解和实施的操作模型（见图1-1），这个模型包含3个部分。

- 部署层

- 配置语言与资源抽象层
- 事务层

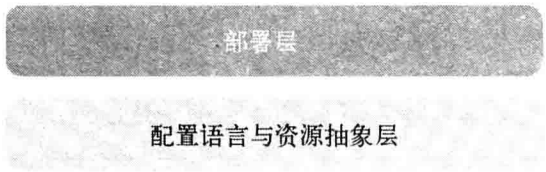

图1-1　Puppet结构模型

1.1.1　部署层

Puppet通常会在简单的“服务器端–客户端”模式下运行（见图1-2）。服务器端称为Puppet master，客户端称为agent，客户端主机称为节点（node）。

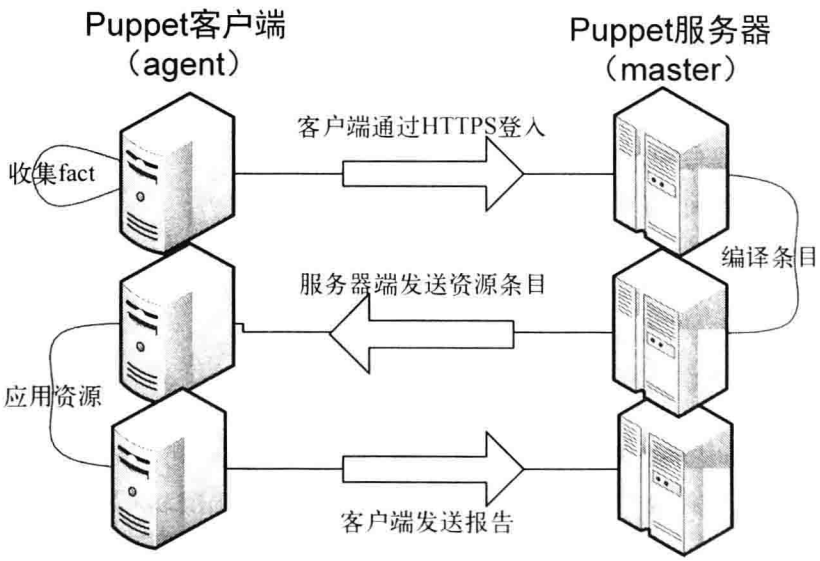

图1-2　Puppet配置的总体运行方式

Puppet master以守护进程的方式运行，包含某一特定环境需要的配置信息，agent通过用标准SSL加密并认证的连接与master建立连接，获取本机需要的配置信息。

需要说明的是，在agent未取得配置信息，或者已经达到配置状态时，Puppet不会对系统进行任何改动。它只有在被要求的时候才修改系统，这是Puppet的一个关键特征，称为幂等性（idempotency），这个修改过程称为一次配置运行（configuration run）。

Puppet agent通常以守护进程方式运行，也可以通过cron运行，或者手工触发。以守护进程运行时，agent周期性地检查master上的配置项是否发生了改变，当有改变发生时获取新的配置信息（见图1-3），也有很多人习惯通过cron或者手工运行。默认情况下Puppet agent每30分钟与master进行一次交互，以

确认配置项是否发生了改变，这个时间间隔可以根据自己的需要灵活调整。

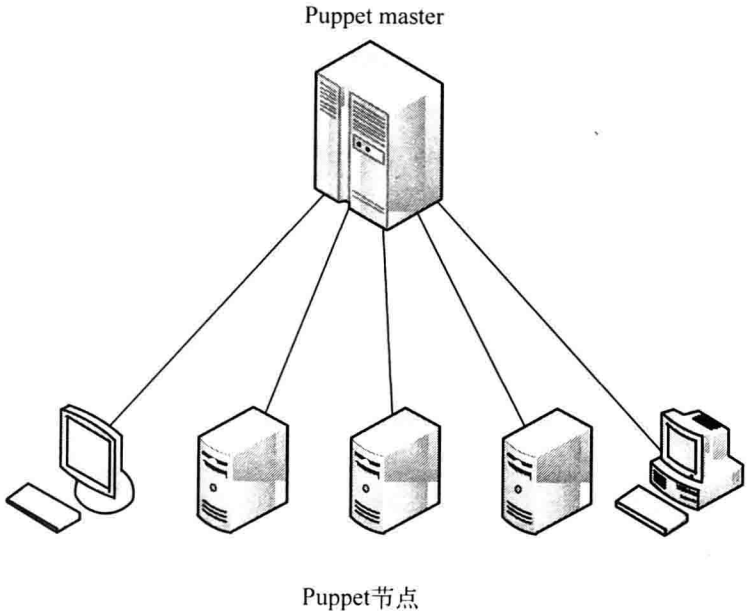

图1-3 Puppet的服务器端–客户端模型

也可以选择其他部署模型，比如独立运行模式，在此模式下不需要Puppet master。配置信息保存在主机本地磁盘上，Puppet程序解析并执行这些配置，我们在第4章讨论这种模式。

1.1.2 配置语言与资源抽象层

Puppet使用一种称为“Puppet语言”的描述性语言定义配置项（Puppet称之为“资源”）。这种“描述性”将Puppet与许多其他配置工具区分开来，所谓“描述性”，是指配置项所处的状态，例如一个包应该被安装，一个服务应该已启动。

大多数配置工具（如Shell或者Perl脚本）是命令式或者叫过程式的，它们描述的是“怎样”达到最终状态而不是最终状态是什么，大多数系统管理员自己编写的脚本都属于过程式之列。

与此相对，Puppet用户只需要描述主机应该达到的最终状态：哪一个包应该被安装，哪一个服务应该已启动，等等。使用Puppet，系统管理员不需要关心最终状态是如何达到的，那是Puppet的事，换句话说，把主机的配置信息抽象为一个个资源。

1. 配置语言

那么这个“描述式语言”到底是怎么回事？让我们通过一个简单的例子说明。假设我们有Red HatEnterprise Linux、Ubuntu和Solaris主机，现在需要在所有主机上安装vim包，如果手工安装，需要以下步骤：

- 连接到某一主机（需要处理密码或者密钥）；
- 检查vim包是否已安装；
- 若没有安装，需要根据具体的系统类型选择合适的命令安装vim包，例如Red Hat系统中使用yum命令，Ubuntu系统中使用apt -get命令；
- 安装结束后需要根据返回的结果确认安装是否成功。

注意　如果希望升级vim（已安装）或者安装某一个指定版本，情况会更加复杂。

Puppet采用了完全不同的方法。使用Puppet，只需要为vim包定义一个配置资源就行了。每个资源包括一个类型（type，被管理资源的种类：包、服务或者定时任务）、一个名称（title，资源的名称），以及一系列属性（attribute，描述资源状态的值，例如某个服务是启动状态还是停止状态）。

代码清单1-1是一个Puppet资源示例。

代码清单1-1　一个Puppet资源

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

代码清单1-1中描述了一个名为vim的包应该被安装，它的结构如下：

```
类型 { 标题:
  属性 => 值,
}
```

代码清单1-1中，资源类型是package（包），Puppet默认支持一些资源类型，如用来管理文件、服务、包和定时任务等的类型。

注意　Puppet支持的完整资源类型（及其属性）列表见http://docs.puppetlabs.com/references/stable/type.html。也可以通过扩展Puppet使它支持更多类型，这部分内容将在第10章中讨论。

接下来是资源名称，这里是我们要安装的包名是vim，与包管理器的参数名称是一致的，如apt-get install vim。

最后，我们指定了一个属性ensure，以及它的值present。属性告诉Puppet该资源需要达到的状态，每个资源都需要一系列属性来确定它的状态。这里的ensure属性用来指定包的状态：已安装、未安装等。值present告诉Puppet我们要求安装这个包，如果要求不安装这个包，需要将属性值改为absent。

2. 资源抽象层

资源创建完成后，当agent连接到master时，Puppet负责处理管理资源的细节工作。Puppet知道不同平台和操作系统上管理某种类型资源的具体方法，每一种资源都有多个提供者（provider），每个提供者通过这个平台自身的包管理工具来完成管理包的具体工作。

以package类型为例，它有超过20个提供者，覆盖了相当多的包管理工具，例如yum、aptitude、pkgadd、ports和emerge等。

当一个agent连接master时，Puppet用一个名为Facter的工具获取agent主机信息（见以下补充内容），例如该主机的操作系统。Puppet根据这一信息选择合适的包提供者，并通过该提供者检查vim包是否已经安装在了目标操作系统上，例如，在Red Hat上执行yum命令，在Ubuntu上执行aptitude命令，在Solaris上执行pkg命令。如果检查结果是未安装包，Puppet将安装此软件包，否则Puppet什么都不做，这就是前面提到的幂等性。

最后Puppet将应用此资源的结果报告（成功或者失败）发回master。

Facter和fact

Facter是一个本书经常用到的系统盘点工具。与Puppet一样，它也是由Puppet Labs开发、以Apache 2.0许可证分发的开源软件。它的功能是报告每个节点的一些fact（事实），如主机名、IP地址、操作系统及版本，以及其他一些配置信息。这些信息是在agent运行时被收集的。这些fact被发送给Puppet master，由后者自动转换为Puppet顶级域中可以使用的变量。我们在第2章中会深入讨论变量域的概念。

在agent节点上，我们可以通过在命令行中运行facter命令获取节点的fact，每个fact以“key => value”对的形式返回：

```
$ facter
operatingsystem => Ubuntu
ipaddress => 10.0.0.10
```

基于这些节点信息，我们可以对每台主机进行单独定制，比如可以根据主机的IP地址配置这台主机的网络。

这些fact可以在Puppet配置文件中作为普通变量使用，结合节点配置文件，可以实现对每台主机定义不同的配置项。例如，你可以定义通用的网络配置，然后根据每个agent返回的具体值定制各自的具体配置实现。

Facter的另一个作用是帮助Puppet确定管理某个节点资源需要使用的工具。例如，当Facter返回fact表明节点主机上运行的是Ubuntu时，Puppet将使用aptitude工具在此节点上安装软件包。我们可以通过扩展Facter工具使它返回某些主机上的用户自定义fact。我们将在安装Puppet后紧接着安装Facter，并在后续章节详细讨论它。

1.1.3 事务层

事务层是Puppet的工作引擎，一个Puppet事务包含配置一台agent主机的完整过程，包括如下步骤：

- 解析并编译配置；
- 将编译后的配置发送到agent上；
- 在agent上应用编译后的配置；
- 将运行结果报告传回master。

下面详细解释这几步。首先，Puppet分析配置信息并计算如何在agent上运行这些配置项。具体方法是：Puppet将所有配置项转换成一个关系图，包含各个资源之间的关系以及与每个agent之间的关系。这种方式使得我们编写配置时不需要关心资源之间的顺序，Puppet能够保证每个资源按照正确的顺序被应用在agent上。这是Puppet最酷的特点之一。

然后，Puppet基于特定agent将配置信息编译为catalog，然后将它发送到此agent所在的主机上，运行结果以报告（report）的形式返回master。

事务层允许配置在主机上被多次创建和应用。由于Puppet的幂等特性，同一操作不论被执行多少次，在agent主机上产生的结果是相同的，这一特点与前述守护进程的工作方式相结合，使得Puppet客户端与服务器端保持持续交互，从而实现了配置信息的一致性。

然而，Puppet并没有实现完整意义上的事务性，事务在Puppet中没有实现原子性，只是用日志文

件进行了记录，所以无法像一些数据库那样对事务进行回滚（rollback，类似于文本编辑中的undo操作）。Puppet解决这个问题的方法是增加了一种noop（即无操作）模式，类似于文本编辑中的预览，你可以在这个模式下观察应用配置将会对系统进行的修改，但不真正应用这些配置，直到你对修改满意，再应用配置。

1.2　选择正确的版本

Puppet最好的版本通常是最新的发布版，本书写作的时候最新发布版是3.2.*x*，更新的版本正在开发中。相比于之前的版本，3.2.*x*版在性能上有了很大提升，并整合了Hiera。Hiera是一个Puppet外部数据存储工具，后续章节会详细介绍它。

3.1.*x*是一个稳定的版本，性能强劲，在之前版本的基础上修复了大量bug，并包含一些旧版本没有的新特性。

注意　本书假设你使用的Puppet版本不低于3.1.*x*，书中的部分内容可以运行在2.7.*x*及之上版本，但全部内容并没有经过一一测试。尤其需要注意的是，书中关于Hiera（见第12章）和函数的内容不适用于2.7.*x*版本。

类Unix操作系统家族，如Linux、Mac OS X等都有自己的软件包仓库，Puppet的一些版本被加入到了这些仓库中，其中2.7.*x*被采用得最多，3.1.*x*只被收进了一些比较新的操作系统版本仓库中。如果你在自己操作系统的包仓库中找不到想要的Puppet版本，可以下载最新的安装包，或者手工从源代码构建Puppet（但我们不推荐后者）。Puppet Labs官网上提供了最新版本Puppet的rpm（适于Red Hat、CentOS、Fedora系列Linux）、deb（适于Debian、Ubuntu系列Linux）、MSI（适于Microsoft Windows）和dmg（适于Max OS X）格式安装包下载。

多版本Puppet混合部署

Puppet最常见的部署方式是服务器端-客户端方式，许多人问是否可以在服务器端和客户端上安装不同版本的Puppet，答案是可以，但有一些注意事项。首先，服务器端上部署的Puppet版本不能低于客户端上的Puppet版本，比如，2.7.20版本的agent可以连接3.1.1版本的master，但3.1.1版本的agent无法连接2.7.20版本的master。

其次，agent的版本越低，它与master连接后正常工作的可能性就越小。新版本的master可能与旧版本的agent不兼容，一些函数和特性不能正常工作。

最后，3.1.*x*及以上版本的master与2.7.*x*及以下版本的agent混合部署，将无法获得完全3.1.*x*版本部署时的性能提升。

1.3　安装 Puppet

Puppet可以安装或使用在多个平台下，其中包括：

- Red Hat Enterprise Linux、CentOS、Fedora和Oracle Enterprise Linux
- Debian和Ubuntu
- OpenIndiana
- Solaris
- 从源代码安装
- Microsoft Windows（仅客户端）
- Mac OS X和Mac OS X Server
- 其他（如BSD、Mandrake和Mandriva等）

在这些平台上Puppet能够管理的配置项包括但不限于：

- 文件
- 服务
- 包
- 用户
- 组
- 定时任务
- SSH密钥
- Nagios配置

虽然大多数平台和发布包管理系统中Puppet的功能被分为agent和master两部分，但二者的安装过程十分类似。在一些系统和发行版上你可能要单独安装Ruby及其库文件，以及一些其他的包，但一些好的软件包管理系统中都将Ruby等包作为Puppet和Facter的依赖包，在安装Puppet的过程中就自动安装了。对于一些其他特性（例如后面要讲到的报表特性），你可能需要单独安装。

后面我们会演示如何从源代码安装Puppet，但不推荐这种方式。使用操作系统的包管理器自动安装通常会比较容易，尤其是你需要在大量主机上安装Puppet时。

1.3.1 Red Hat Enterprise Linux和Fedora

把Red Hat Enterprise Linux扩展包仓库（EPEL）或者Puppet Labs软件包仓库加入到主机的软件包源列表中，就可以在线安装Puppet了。注意，本书写作的时候，只能通过Puppet Labs软件仓库安装Puppet 3。

1. 安装EPEL仓库

EPEL仓库是一个由Fedora项目志愿者发起并维护的高质量扩展包仓库，为Red Hat Enterprise Linux及其衍生版本如CentOS、Oracle Enterprise Linux和Scientific Linux提供扩展支持。在http://fedoraproject.org/wiki/EPEL和http://fedoraproject.org/wiki/EPEL/FAQ#howtouse上可以找到更多使用EPEL（包括如何在你的主机上添加EPEL）的帮助。

使用如下方法将epel-release RPM（.rpm包管理器）加入到主机包更新源中。

- Enterprise Linux 5：

```
# rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
```

- Enterprise Linux 6：

```
# rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
```

2. 安装Puppet Labs仓库

用类似的方法在Enterprise Linux 5和6上安装Puppet Labs仓库。

- Enterprise Linux 5：

```
# rpm -ivh http://yum.puppetlabs.com/el/5/products/i386/puppetlabs-release-5-7.noarch.rpm
```

- Enterprise Linux 6：

```
# rpm -ivh http://yum.puppetlabs.com/el/6/products/i386/puppetlabs-release-6-7.noarch.rpm
```

3. 安装EPEL和Puppet Labs软件包

Puppet master上需要从EPEL或者Puppet Labs软件仓库中安装puppet、puppet-server和facter包：

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

其中puppet包包含agent，puppet-server包包含master，facter包包含系统盘点工具Facter。前面已经讲过，Facter通过收集系统信息（称为fact）帮助Puppet定制系统配置。

在agent上只要安装puppet和facter包就可以了。

```
# yum install puppet facter
```

4. 通过RubyGems安装

与大多数基于Ruby的应用相同，Puppet和Facter也可以通过RubyGems安装。这种方式需要事先在操作系统上安装Ruby和对应的RubyGems包。在Red Hat、CentOS、Fedora、SUSE/SLES、Debian和Ubuntu上，这个包的名字是rubygems。安装完RubyGems后就可以使用gem命令来安装Puppet和Facter了：

```
# gem install puppet facter
```

1.3.2　Debian和Ubuntu

在Debian和Ubuntu系统中，puppet包和puppetmaster包分别包含Puppet agent和master。在master上需要执行：

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

在agent上需要执行：

```
# apt-get install puppet
```

注意　安装puppet、puppetmaster和facter时，系统会自动安装它们的依赖包。例如，如果系统中没有Ruby，会自动安装它。

最新版本的Puppet用下面的方法添加Puppetlabs软件仓库。

- Debian Wheezy：

```
# wget http://apt.puppetlabs.com/puppetlabs-release-wheezy.deb
# dpkg -i puppetlabs-release-wheezy.deb
# apt-get update
```

- Ubuntu Precise：