

Docker技术入门与实战（第2版）(容器技术系列)

作者：杨保华

容器技术系列

Docker技术入门与实战（第2版）

杨保华 戴王剑 曹亚仑 编著

ISBN: 978-7-111-55582-7

本书纸版由机械工业出版社于2017年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

目录
第2版前言
第1版前言
第一部分 基础入门
第1章 初识容器与Docker
1.1 什么是Docker
1.2 为什么要使用Docker
1.3 Docker与虚拟化
1.4 本章小结
第2章 核心概念与安装配置
2.1 核心概念
2.2 安装Docker
2.3 配置Docker服务
2.4 推荐实践环境
2.5 本章小结
第3章 使用Docker镜像
3.1 获取镜像
3.2 查看镜像信息
3.3 搜寻镜像
3.4 删除镜像
3.5 创建镜像
3.6 存出和载入镜像
3.7 上传镜像
3.8 本章小结
第4章 操作Docker容器
4.1 创建容器
4.2 终止容器
4.3 进入容器
4.4 删除容器
4.5 导入和导出容器
4.6 本章小结
第5章 访问Docker仓库
5.1 Docker Hub公共镜像市场
5.2 时速云镜像市场
5.3 搭建本地私有仓库
5.4 本章小结
第6章 Docker数据管理
6.1 数据卷
6.2 数据卷容器
6.3 利用数据卷容器来迁移数据
6.4 本章小结
第7章 端口映射与容器互联
7.1 端口映射实现访问容器
7.2 互联机制实现便捷互访
7.3 本章小结
第8章 使用Dockerfile创建镜像
8.1 基本结构
8.2 指令说明
8.3 创建镜像
8.4 使用.dockerignore文件
8.5 最佳实践
8.6 本章小结
第二部分 实战案例
第9章 操作系统
9.1 BusyBox
9.2 Alpine
9.3 Debian/Ubuntu
9.4 CentOS/Fedora
9.5 本章小结
第10章 为镜像添加SSH服务
10.1 基于commit命令创建
10.2 使用Dockerfile创建
10.3 本章小结
第11章 Web服务与应用
11.1 Apache
11.2 Nginx
11.3 Tomcat
11.4 Jetty
11.5 LAMP
11.6 CMS
11.7 持续开发与管理
11.8 本章小结
第12章 数据库应用
12.1 MySQL
12.2 MongoDB
12.3 Redis
12.4 Memcached
12.5 CouchDB
12.6 Cassandra
12.7 本章小结
第13章 分布式处理与大数据平台
13.1 RabbitMQ
13.2 Celery
13.3 Hadoop
13.4 Spark
13.5 Storm
13.6 Elasticsearch
13.7 本章小结
第14章 编程开发
14.1 C/C++
14.2 Java
14.3 Python
14.4 JavaScript
14.5 Go
14.6 PHP
14.7 Ruby
14.8 Perl

- [14.9 R](#)
- [14.10 Erlang](#)
- [14.11 本章小结](#)
- [第15章 容器与云服务](#)
- [15.1 公有云容器服务](#)
- [15.2 容器云服务](#)
- [15.3 阿里云容器服务](#)
- [15.4 时速云容器平台](#)
- [15.5 本章小结](#)
- [第16章 容器实战思考](#)
- [16.1 Docker为什么会成功](#)
- [16.2 研发人员该如何看容器](#)
- [16.3 容器化开发模式](#)
- [16.4 容器与生产环境](#)
- [16.5 本章小结](#)
- [第三部分 进阶技能](#)
- [第17章 Docker核心实现技术](#)
- [17.1 基本架构](#)
- [17.2 命名空间](#)
- [17.3 控制组](#)
- [17.4 联合文件系统](#)
- [17.5 Linux网络虚拟化](#)
- [17.6 本章小结](#)
- [第18章 配置私有仓库](#)
- [18.1 安装Docker Registry](#)
- [18.2 配置TLS证书](#)
- [18.3 管理访问权限](#)
- [18.4 配置Registry](#)
- [18.5 批量管理镜像](#)
- [18.6 使用通知系统](#)
- [18.7 本章小结](#)
- [第19章 安全防护与配置](#)
- [19.1 命名空间隔离的安全](#)
- [19.2 控制组资源控制的安全](#)
- [19.3 内核能力机制](#)
- [19.4 Docker服务端的防护](#)
- [19.5 更多安全特性的使用](#)
- [19.6 使用第三方检测工具](#)
- [19.7 本章小结](#)
- [第20章 高级网络功能](#)
- [20.1 网络启动与配置参数](#)
- [20.2 配置容器DNS和主机名](#)
- [20.3 容器访问控制](#)
- [20.4 映射容器端口到宿主主机的实现](#)
- [20.5 配置docker0网桥](#)
- [20.6 自定义网桥](#)
- [20.7 使用OpenvSwitch网桥](#)
- [20.8 创建一个点到点连接](#)
- [20.9 本章小结](#)
- [第21章 libnetwork插件化网络功能](#)
- [21.1 容器网络模型](#)
- [21.2 Docker网络相关命令](#)
- [21.3 构建跨主机容器网络](#)
- [21.4 本章小结](#)
- [第四部分 开源项目](#)
- [第22章 Etcd——高可用的键值数据库](#)
- [22.1 简介](#)
- [22.2 安装和使用Etcd](#)
- [22.3 使用etcdctl客户端](#)
- [22.4 Etcd集群管理](#)
- [22.5 本章小结](#)
- [第23章 Docker三剑客之Docker Machine](#)
- [23.1 简介](#)
- [23.2 安装Machine](#)
- [23.3 使用Machine](#)
- [23.4 Machine命令](#)
- [23.5 本章小结](#)
- [第24章 Docker三剑客之Docker Compose](#)
- [24.1 简介](#)
- [24.2 安装与卸载](#)
- [24.3 Compose命令说明](#)
- [24.4 Compose环境变量](#)
- [24.5 Compose模板文件](#)
- [24.6 Compose应用案例一：Web负载均衡](#)
- [24.7 Compose应用案例二：大数据Spark集群](#)
- [24.8 本章小结](#)
- [第25章 Docker三剑客之Docker Swarm](#)
- [25.1 简介](#)
- [25.2 安装Swarm](#)
- [25.3 使用Swarm](#)
- [25.4 使用其他服务发现后端](#)
- [25.5 Swarm中的调度器](#)
- [25.6 Swarm中的过滤器](#)
- [25.7 本章小结](#)
- [第26章 Mesos——优秀的集群资源调度平台](#)
- [26.1 简介](#)
- [26.2 Mesos安装与使用](#)
- [26.3 原理与架构](#)
- [26.4 Mesos配置项解析](#)
- [26.5 日志与监控](#)
- [26.6 常见应用框架](#)
- [26.7 本章小结](#)
- [第27章 Kubernetes——生产级容器集群平台](#)
- [27.1 简介](#)
- [27.2 核心概念](#)
- [27.3 快速体验](#)
- [27.4 安装部署](#)

[27.5 重要组件](#)
[27.6 使用kubectl](#)
[27.7 网络设计](#)
[27.8 本章小结](#)
[第28章 其他相关项目](#)
[28.1 平台即服务方案](#)
[28.2 持续集成平台Drone](#)
[28.3 容器管理](#)
[28.4 编程开发](#)
[28.5 网络支持](#)
[28.6 日志处理](#)
[28.7 服务代理工具](#)
[28.8 标准与规范](#)
[28.9 其他项目](#)
[28.10 本章小结](#)
[附录](#)
[附录A 常见问题总结](#)
[附录B Docker命令查询](#)
[附录C 参考资源链接](#)

第2版前言

自云计算步入市场算起，新一代计算技术刚好走过了第一个十年。

在过去十年里，围绕计算、存储、网络三大基础服务，围绕敏捷服务和规模处理两大核心诉求，新的概念、模式和工具争相涌现。这些创新的开源技术成果，提高了整个信息产业的生产效率，降低了应用信息技术的门槛，让“互联网+”成为可能。

如果说软件定义网络（SDN）和网络功能虚拟化（NFV）让互联网络的虚拟化进入了崭新的阶段，那么容器技术的出现，毫无疑问称得上计算虚拟化技术的又一大创新。从Linux Container到Docker，看似是计算技术发展的一小步，却是极为重要的历史性突破。容器带来的不仅仅是技术体验上的改进，更多的是新的开发模式、新的应用场景、新的业务可能……

容器技术自身在快速演进的同时，笔者也很欣喜地看到，围绕着容器的开源生态系统越发繁盛。Docker三剑客Machine、Compose、Swarm相辅相成，集团作战；搜索巨人则推出Kubernetes，领航新一代容器化应用集群平台；还有Mesos、CoreOS，以及其他众多的开源工具。这些工具的出现，弥补了现有容器技术栈的不足，极大地丰富了容器技术的应用场景，增强了容器技术在更多领域的竞争力。

在第2版中，笔者参照容器技术最新进展对全书内容进行了修订完善，并增加了第四部分专门介绍与容器相关的知名开源项目，利用好这些优秀的开源平台，可以更好地在生产实践中受益。

成书之际，Docker发布了1.13版本，带来了更稳定的性能和更多有趣的特性。

再次感谢容器技术，感谢开源文化，希望开源技术能得到更多的支持和贡献！

最后，IBM中国研究院的刘天成、李玉博等帮忙审阅了部分内容，在此表达最深厚的感谢！

杨保华

2016年12月于北京

第1版前言

在一台服务器上同时运行一百个虚拟机，肯定会被认为是痴人说梦。而在一台服务器上同时运行一千个Docker容器，这已经成为现实。在计算机技术高速发展的今天，昔日的天方夜谭正在一个个变成现实。

多年的研发和运维（DevOps）经历中，笔者时常会碰到这样一个困境：用户的需求越来越多样，系统的规模越来越庞大，运行的软件越来越复杂，环境配置问题所造成的麻烦层出不穷……为了解决这些问题，开源社区推出过不少优秀的工具。这些方案虽然在某些程度上确能解决部分“燃眉之急”，但是始终没有一种方案能带来“一劳永逸”的效果。

让作为企业最核心资源的工程师们花费大量的时间，去解决各种环境和配置引发的Bug，这真的正常吗？

回顾计算机的发展历程，最初，程序设计人员需要直接操作各种枯燥的机器指令，编程效率之低可想而知。高级语言的诞生，将机器指令的具体实现成功抽象出来，从此揭开了计算机编程效率突飞猛进的大时代。那么，为什么不能把类似的理念（抽象与分层）也引入到现代的研发和运维领域呢？

Docker无疑在这一方向上迈出了具有革新意义的一步。笔者在刚接触Docker时，就为它所能带来的敏捷工作流程而深深吸引，也为它能充分挖掘云计算资源的效能而兴奋不已。我们深信，Docker的出现，必将给DevOps技术，甚至整个信息技术产业的发展带来深远的影响。

笔者曾尝试编写了介绍Docker技术的中文开源文档。短短一个月的时间，竟收到了来自全球各个地区超过20万次的阅读量和全五星的好评。这让我们看到国内技术界对于新兴开源技术的敏锐嗅觉和迫切需求，同时也倍感压力，生怕其中有不妥之处，影响了大家学习和推广Docker技术的热情。在开源文档撰写过程中，我们一直在不断思考，在生产实践中到底怎么用Docker才是合理的？在“华章图书”的帮助下，终于有了现在读者手中的这本书。

与很多技术类书籍不同，本书中避免一上来就讲述冗长的故事，而是试图深入浅出、直奔主题，在最短时间内让读者理解和掌握最关键的技术点，并且配合实际操作案例和精炼的点评，给读者提供真正可以上手的实战指南。

本书在结构上分为三大部分。第一部分是Docker技术的基础知识介绍，这部分将让读者对Docker技术能做什么有个全局的认识；第二部分将具体讲解各种典型场景的应用案例，供读者体会Docker在实际应用中的高效秘诀；第三部分将讨论一些偏技术环节的高级话题，试图让读者理解Docker在设计上的工程美学。最后的附录归纳了应用Docker的常见问题和一些常用的参考资料。读者可根据自身需求选择阅读重点。全书主要由杨保华和戴王剑主笔，曹亚仑写作了编程开发和实践之道章节。

本书在写作过程中参考了官方网站上的部分文档，并得到了DockerPool技术社区网友们的积极反馈和支持，在此一并感谢！

成稿之际，Docker已经发布了增强安全特性的1.3.2版本。衷心祝愿Docker及相关技术能够快速成长和成熟，让众多IT从业人员的工作和生活都更加健康、更加美好！

作者于2014年11月

第一部分 基础入门

·第1章 初识容器与Docker

·第2章 核心概念与安装配置

·第3章 使用Docker镜像

·第4章 操作Docker容器

·第5章 访问Docker仓库

·第6章 Docker数据管理

·第7章 端口映射与容器互联

·第8章 使用Dockerfile创建镜像

本部分共有8章内容，笔者将介绍Docker和容器的相关基础知识。

第1章介绍Docker的前世与今生，以及它与现有的虚拟化技术，特别是Linux容器技术的关系。

第2章介绍Docker的三大核心概念，以及如何在常见的操作系统环境中安装Docker。

第3章到第5章通过具体的示例，讲解使用Docker的常见操作，包括镜像、容器和仓库。

第6章剖析如何在Docker中使用数据卷来保存持久化数据。

第7章介绍如何使用端口映射和容器互联来方便外部对容器服务的访问。

第8章介绍如何编写Dockerfile配置文件，以及使用Dockerfile来创建镜像的具体方法和注意事项。

第1章 初识容器与Docker

如果说主机时代大家比拼的是单个服务器物理性能（如CPU主频和内存），那么在云时代，最为看重的则是凭借虚拟化技术所构建的集群处理能力。

伴随着信息技术的飞速发展，虚拟化技术早已经广泛应用到各种关键场景中。从20世纪60年代IBM推出的大型主机虚拟化，到后来以Xen、KVM为代表的虚拟机虚拟化，再到现在以Docker为代表的容器技术，虚拟化技术自身也在不断进行创新和突破。

传统来看，虚拟化既可以通过硬件模拟来实现，也可以通过操作系统软件来实现。而容器技术则更为优雅，它充分利用了操作系统本身已有的机制和特性，可以实现远超传统虚拟机的轻量级虚拟化。因此，有人甚至把它称为“新一代的虚拟化”技术，并将基于容器打造的云平台亲切地称为“容器云”。

Docker毫无疑问正是众多容器技术中的佼佼者，是容器技术发展过程中耀眼的一抹亮色。那么，什么是Docker？它会带来哪些好处？它跟现有虚拟化技术又有何关系？

本章首先会介绍Docker项目的起源和发展过程，之后会为大家剖析Docker和相关容器技术，以及它在DevOps等场景带来的巨大便利。最后，还将阐述Docker在整个虚拟化领域中的技术定位。

1.1 什么是Docker

1.Docker开源项目背景

Docker是基于Go语言实现的开源容器项目，诞生于2013年年初，最初发起者是dotCloud公司。Docker自开源后受到广泛的关注和讨论，目前已有多个相关项目（包括Docker三剑客、Kubernetes等），逐渐形成了围绕Docker容器的生态体系。

由于Docker在业界造成的影响力实在太太大，dotCloud公司后来也直接改名为Docker Inc，并专注于Docker相关技术和产品的开发。

。

图1-1 Docker官方网站

Docker项目已加入了Linux基金会，并遵循Apache2.0协议，全部开源代码均在<https://github.com/docker/docker>上进行维护。在Linux基金会最近一次关于“最受欢迎的云计算开源项目”的调查中，Docker仅次于2010年发起的OpenStack项目，并仍处于上升趋势。

现在主流的Linux操作系统都已经支持Docker。例如，红帽公司的RHEL 6.5/CentOS 6.5往上的操作系统、Ubuntu 14.04往上的操作系统，都已经在软件源中默认带有Docker软件包。Google公司宣称在其PaaS（Platform as a Service）平台及服务产品中广泛应用了Docker容器。IBM公司跟Docker公司达成了战略合作伙伴关系。微软公司在其云平台Azure上加强了对Docker的支持。公有云提供商亚马逊也推出了AWS EC2 Container服务，提供对Docker和容器业务的支持。

Docker的构想是要实现“Build, Ship and Run Any App, Anywhere”，即通过对应用的封装（Packaging）、分发（Distribution）、部署（Deployment）、运行（Runtime）生命周期进行管理，达到应用组件“一次封装，到处运行”的目的。这里的应用组件，既可以是一个Web应用、一个编译环境，也可以是一套数据库平台服务，甚至是一个操作系统或集群。

基于Linux平台上的多项开源技术，Docker提供了高效、敏捷和轻量级的容器方案，并支持部署到本地环境和多种主流云平台。可以说，Docker首次为应用的开发、运行和部署提供了“一站式”的实用解决方案。

2.Linux容器技术——巨人的肩膀

跟大部分新兴技术的诞生一样，Docker也并非“从石头缝里蹦出来的”，而是站在前人的肩膀上，其中最重要的就是Linux容器（Linux Containers，LXC）技术。

IBM DeveloperWorks网站关于容器技术的描述十分准确：“容器有效地将由单个操作系统管理的资源划分到孤立的组中，以更好地在孤立的组之间平衡有冲突的资源使用需求。与虚拟化相比，这样既不需要指令级模拟，也不需要即时编译。容器可以在核心CPU本地运行指令，而不需要任何专门的解释机制。此外，也避免了准虚拟化（paravirtualization）和系统调用替换中的复杂性。”

当然，LXC也经历了长期的演化。最早的容器技术可以追溯到1982年Unix系列操作系统上的chroot工具（直到今天，主流的Unix、Linux操作系统仍然支持和带有该工具）。早期的容器实现技术包括Sun Solaris操作系统上的Solaris Containers（2004年发布），FreeBSD操作系统上的FreeBSD jail（2000年左右出现），以及GNU/Linux上的Linux-VServer和OpenVZ。

在LXC之前，这些相关技术经过多年的演化已经十分成熟和稳定，但是由于种种原因，它们并没有被很好地集成到主流的Linux内核中，用户使用起来并不方便。例如，如果用户要使用OpenVZ技术，需要先手动给操作系统打上特定的内核补丁方可使用，而且不同版本并不一致。类似的困难造成在很长一段时间内，这些优秀的技术只流传于技术人员的小圈子中。

后来LXC项目借鉴了前人成熟的容器设计理念，并基于一系列新引入的内核特性，实现了更具扩展性的虚拟化容器方案。更加关键的是，LXC终于被集成到了主流Linux内核中，进而成为了Linux系统轻量级容器技术的事实标准。从技术层面来看，LXC已经越过了绝大部分的“坑”，完成了容器技术实用化的大半历程。

3.从Linux容器到Docker

在LXC的基础上，Docker进一步优化了容器的使用体验，让它进入了寻常百姓家。

首先，Docker提供了各种容器管理工具（如分发、版本、移植等）让用户无需关注底层的操作，可以更简单地管理和使用容器；其次，Docker通过引入分层文件系统构建和高效的镜像机制，降低了迁移难度，极大地提升了用户体验。用户操作Docker容器就像操作应用自身一样简单。

早期的Docker代码实现是直接基于LXC的。自0.9版本开始，Docker开发了libcontainer项目，作为更广泛的容器驱动实现，从而替换掉了LXC的实现。目前，Docker还积极推动成立了runC标准项目，试图让容器支持不再局限于Linux操作系统，而是更安全、更具扩展性。

简单地讲，读者可以将Docker容器理解为一种轻量级的沙盒（sandbox）。每个容器内运行着一个应用，不同的容器相互隔离，容器之间也可以通过网络互相通信。容器的创建和停止都十分快速，几乎跟创建和终止原生应用一致；另外，容器自身对系统资源的额外需求也十分有限，远远低于传统虚拟机。很多时候，甚至直接把容器当作应用本身也没有任何问题。

有理由相信，Docker技术会进一步成熟，将成为更受欢迎的容器虚拟化技术实现，并在云计算和DevOps等领域得到更广泛的应用。

1.2 为什么要使用Docker

1.Docker容器虚拟化的好处

Docker项目的发起人和Docker公司CTO Solomon Hykes曾认为，Docker在正确的地点、正确的时间顺应了正确的趋势——如何正确地构建应用。

在云时代，开发者创建的应用必须要能很方便地在网络上传播，也就是说应用必须脱离底层物理硬件的限制；同时必须是“任何时间、任何地点”可获取的。因此，开发者需要一种新型的创建分布式应用程序的方式，快速分发和部署，这正是Docker所能够提供的最大优势。

举个简单的例子，假设用户试图基于最常见的LAMP（Linux+Apache+MySQL+PHP）组合来构建一个网站。按照传统的做法，首先，需要安装Apache、MySQL和PHP以及它们各自运行所依赖的环境；之后分别对它们进行配置（包括创建合适的用户、配置参数等）；经过大量的操作后，还需要进行功能测试，看是否工作正常；如果不正常，则进行调试追踪，意味着更多的时间代价和不可控的风险。可以想象，如果应用数目变多，事情会变得更加难以处理。

更为可怕的是，一旦需要服务器迁移（例如从亚马逊云迁移到其他云），往往需要对每个应用都进行重新部署和调试。这些琐碎而无趣的“体力活”，极大地降低了工作效率。究其根源，是这些应用直接运行在底层操作系统上，无法保证同一份应用在不同的环境中行为一致。

而Docker提供了一种更为聪明的方式，通过容器来打包应用，解耦应用和运行平台。意味着迁移的时候，只需要在新的服务器上启动需要的容器就可以了，无论新旧服务器是否是同一类型的平台。这无疑将节约大量的宝贵时间，并降低部署过程出现问题的风险。

2.Docker在开发和运维中的优势

对开发和运维（DevOps）人员来说，可能最梦寐以求的效果就是一次创建或配置，之后可以在任意地方、任意时间让应用正常运行。而Docker恰恰是可以实现这一终极目标的“瑞士军刀”。

具体说来，Docker在开发和运维过程中，具有如下几个方面的优势：

·更快速的交付和部署。使用Docker，开发人员可以使用镜像来快速构建一套标准的开发环境；开发完成之后，测试和运维人员可以直接使用完全相同环境来部署代码。只要开发测试过的代码，就可以确保在生产环境无缝运行。Docker可以快速创建和删除容器，实现快速迭代，大量节约开发、测试、部署的时间。并且，整个过程全程可见，使团队更容易理解应用的创建和工作过程。

·更高效的资源利用。Docker容器的运行不需要额外的虚拟化管理程序（Virtual Machine Manager, VMM, 以及Hypervisor）支持，它是内核级的虚拟化，可以实现更高的性能，同时对资源的额外需求很低。跟传统虚拟机方式相比，要提高一到两个数量级。

·更轻松的迁移和扩展。Docker容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等，同时支持主流的操作系统发行版本。这种兼容性让用户可以在不同平台之间轻松地迁移应用。

·更简单的更新管理。使用Dockerfile，只需要小小的配置修改，就可以替代以往大量的更新工作。并且所有修改都以增量的方式被分发和更新，从而实现自动化并且高效的容器管理。

3.Docker与虚拟机比较

作为一种轻量级的虚拟化方式，Docker在运行应用上传统的虚拟机方式相比具有显著优势：

·Docker容器很快，启动和停止可以在秒级实现，而传统的虚拟机方式需要数分钟。

·Docker容器对系统资源需求很少，一台主机上可以同时运行数千个Docker容器（在IBM服务器上已经实现了同时运行10K量级的容器实例）。

·Docker通过类似Git设计理念的操作来方便用户获取、分发和更新应用镜像，存储复用，增量更新。

·Docker通过Dockerfile支持灵活的自动化创建和部署机制，提高工作效率，使流程标准化。

Docker容器除了运行其中应用外，基本不消耗额外的系统资源，保证应用性能的同时，尽量减小系统开销。传统虚拟机方式运行N个不同的应用就要起N个虚拟机（每个虚拟机需要单独分配独占的内存、磁盘等资源），而Docker只需要启动N个隔离的“很薄的”容器，并将应用放进容器内即可。应用获得的是接近原生的运行性能。

当然，在隔离性方面，传统的虚拟机方式提供的是相对封闭的隔离。但这并不意味着Docker就不安全，Docker利用Linux系统上的多种防护技术实现了严格的隔离可靠性，并且可以整合众多安全工具。从1.3.0版本开始，Docker重点改善了容器的安全控制和镜像的安全机制，极大提高了使用Docker的安全性。在已知的大规模应用中，目前尚未出现值得担忧的安全隐患。

表1-1总结了使用Docker容器技术与传统虚拟机技术的特性比较，可见容器技术在很多应用场景下都具有巨大的优势。

表1-1 Docker容器技术与传统虚拟机技术的特性比较

□

欢迎访问：电子书学习和下载网站 (<https://www.shgis.cn>)

文档名称：《Docker技术入门与实战（第2版）（容器技术系列）》杨保华 著.epub

请登录 <https://shgis.cn/post/270.html> 下载完整文档。

手机端请扫码查看：

