# 基于 BASH 脚本的 Unix 环境下多组件部署管理框架①

闫生超

(国网电力科学研究院/南京南瑞集团公司, 南京 210003)

**摘 要**: 利用 Unix 环境下广泛使用的 bash/shell 脚本语言, 结合 awk 等内嵌功能命令, 通过合理的文件目录组织和功能实现, 构建成在 Unix 环境下基于 BASH 脚本的多组件部署及管理应用框架, 支持安装、运行、日志、监视、备份、自动升级、清理、帮助等日常操作, 所实现的框架具有良好的可操作性、可扩展性和兼容性, 为 Unix 下多组件的部署提供了良好的应用基础.

**关键词**: Unix; Linux; Shell; Bash; 部署; 管理; 框架; 目录组织

## BASH Scripts Based Deployment and Management Framework for Multi-component Application in Unix Environment

YAN Sheng-Chao

(State Grid Electric Power Research Institute, Nanjing 210003, China)

**Abstract**: By using the widely applied bash/shell scripting languages and making flexible use of awk and other embedded function commands, we managed to construct the deployment and management framework based on the BASH scripts for multi-component in the Unix environment, which claims certain file directory organization. The framework support the day-to-day operations such as installation, operation, logs, monitoring, backup, automatic upgrades, cleaning, helping in its good operability, scalability and compatibility, which provides a good basis for the applications in Unix Environment.

**Key words**: Unix; Linux; shell; Bash; installment; management; framework; directory organization

Unix/Linux 操作系统, 因其具有较强的稳定性、健壮性, 已广泛应用于各类企业级的运行计算环境当中. 但是, 作为支持日常部署和管理的 GUI 图形界面的功能却相对有限, 一方面, 为获得较强的系统稳定性, 其将内核和图形界面隔离开来, 只提供 shell 外壳命令界面, 导致 GUI 的功能反战相对较为滞后, 另外一方面, 因系统使用者对 Unix 的掌握需要一个相对较长的过程, 使得 Unix 下的部署和管理长期以来是一件颇为棘手的事情.

Shell(sh)是提供用户与 Unix、Linux 操作系统之间交互的特殊程序, 第一个主流的 shell 是 Bourne shell(以下简称为 bsh), 以发明者 Steven Bourne 的姓来命名[1], bash 的内置命令相对于 bsh 而言是新增的, 有些是对 bsh 内置命令的改进或继承[2], 具备更为强大的功能, 已防范应用于 Linux 操作系统中, 作为默认的脚本语言出现. 因 bash 这种语言具备较强的灵活性, 及其强大的文件操作、字符解析功能, 因此广泛应用于板件功能测试[3]、文本字频分析[4]、空中交通自动化系统[5]等各个领域.

本文将介绍如何利用 bash 脚本语言, 结合 awk、grep、sed 等内嵌功能命令, 通过合理的文件目录组织和功能实现, 构建成在 Unix 环境下基于 BASH 脚本的多组件部署及管理应用框架, 支持界面友好的组件安装、运行、日志、监视、备份、自动升级、清理、帮助等日常操作, 力图为 Unix 下多组件的部署提供了良好的应用基础.

① 收稿时间:2012-02-25;收到修改稿时间:2012-03-26

## 1　组件部署及管理的需求

长期以来, Unix/Linux 系统广泛应用于 IBM/HP 等小型机上, 几乎独占了其所有操作系统市场份额, 以至于当说到企业级服务器, 第一反应, 必然是 Unix 操作系统. 至于 bash/shell 脚本在系统安装测试[6]服务器状态监测[7]等方面, 已有一些较为成功的实践, 如文献[6-7].

但是, 从个别的实践中获得了一些效果明显不够, 需要通过分析总结, 分析出在 Unix 下的企业级多组件的应用, 在日常部署和管理方面到底有哪些重要的需求, 显得尤为重要. 图 1 为 Unix 下部署及管理需求分析 UML 用例图.

总体而言, 用例角色可分为服务器管理员以及服务器组件安装和自启动脚本等 3 个角色, 用例组包括 5 个, 依次为: 维护服务器组件程序包、备份及还原服务器组件、监视服务器组件运行状态、得到相关帮助信息以及管理服务器组件运行, 以及相应的 26 个衍生用例, 它们的组织关系如图 1 所示.

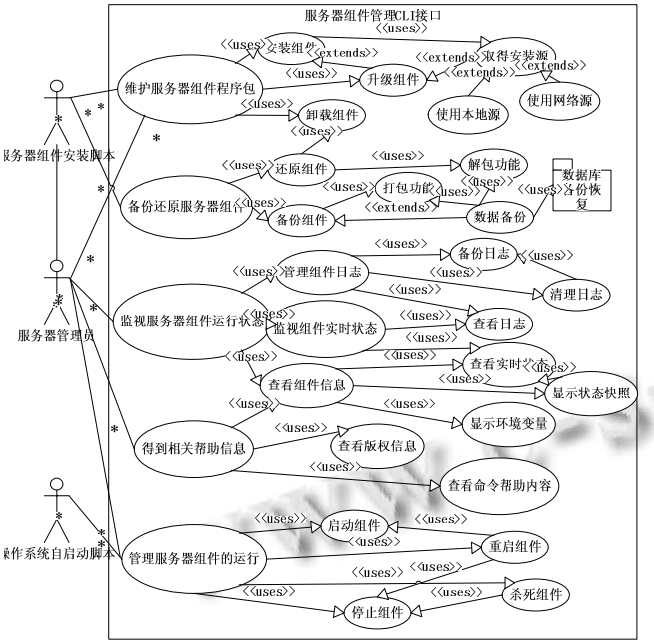


图 1　Unix 下部署及管理需求分析 UML 用例图

## 2　框架的设计及实现

### 2.1　设计的原则方法

文献[8]研究了“自底向上进行 Shell 脚本编程”的理论, 认为需掌握从“外部系统环境”到“内部执行模式”全面的软件方法. 本文所述得到框架系统, 充分遵守这一原则, 采用合理的组织结构, 实现自底向上的 Shell 脚本编程方法, 在实践过程中有效提高脚本的执行效率, 实现了可用性与效率的有效结合. 另外, 在日常部署及维护过程中, 难免会设计到文件的读取与生成, 充分参考了文献[9]所介绍的 UNIX 系统下 Shell 编程技术及其在对文件操作方面的应用, 此外还包括内嵌于 bash 的 awk(字符拆分)、grep(字符查找)、sed(字符重组)、function(函数体)、tar(压缩及解压工具), 请参考具体的 bash 教程.

### 2.2　框架系统的结构

结合需求的分析, 系统的结构如下图 2 所示, 总体上, 从下到上分为 6 层结构, 包括: 系统层、执行层、组件层、功能层、界面层和应用层, 用户的各类操作通过各层之间的数据流动来实现, 各层的功能划分如下:

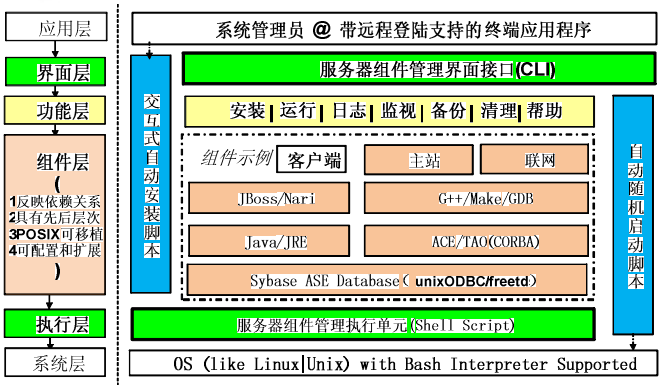


图 2　框架结构图

(1)系统层, 即是支持 BASH 脚本语言运行的操作系统, 包括常见的 Unix 发布版本 AIX、HP-UX, 以及 Linux 发布版本 Redhat、Debian 等;

(2)执行层, 指的是 BASH 脚本语言, 是支持整个框架运行的执行体, Linux 一般内置 bash 执行体, 但对于 Unix 操作系统, 有可能需要支持 bash 运行的程序包, 以支持执行层的具体运行;

(3)组件层, 这一层包括运行的各类组件, 因各个具体的应用不同, 所设计的组件也是多样的, 如图中虚线框所圈选的, 是某个具体的应用, 详情期间后部章节的具体介绍;

(4)功能层, 包括安装、运行、日志、监视、备份还原、清理及帮助等各个具体的功能, 这些框架提供的功能是整个框架重心, 为用户提供了各类应用的封装, 如下:

功能, 命令格式及功能描述

① 安装. 支持组件安装、卸载、升级和更新. 用

法: nix {install|uninstall|upgrade|update} [%mod%]

② 运行. 支持组件的启动、停止、强制退出和重启, 用法: nix {start|stop|kill|restart} [%mod%]

③ 日志. 支持组件、安装、更新日志的显示、备份、清理和清除, 用法: nix log [{show|backup|clean|clear [%mod%]} |install|update]

④ 监视. 支持对组件运行环境和运行情况的显示、监视、统计, 用法: nix {show|watch|status|env|name|info} [%mod%]

⑤ 备份. 支持对文件的压缩和解压、对组件的备份和恢复,用法: nix {tar|untar|backup|restore} [%mod%]

⑥ 清理. 支持对组件的配置、删除和中间文件的清理,用法: nix {confsrv|purge|clean}

⑦ 帮助. 支持对框架的版权、组件的帮助显示, 用法: nix {copyright|help [%mod%]}

(5)界面层, 支持通过命令行界面 CLI, 一般而言既是 shell 或者 X 重大终端;

(6)应用层, 是与用户直接相关的应用抽象.

框架具备两大特色: (1)交互式自动安装, 通过界面层提供的交互界面来引导组件的自动安装, 并在界面层提供倒计时人工干预的自动化安装; (2)自动随系统启动, 通过系统层引用功能层的应用来执行组件层的启动.

### 2.3 框架的功能结构

框架功能结构如上节所示, 框架提供安装、运行、日志、监视、备份、清理及帮助, 包括其具体用法和参数. 其中, %mod%可不带, 则意味对所有组件(all)批量操作; 增删模块: 可依照固定格式发布包的形式实现, 对于命令是透明的; 如此可见, 服务器组件的可高度定制和扩展的.

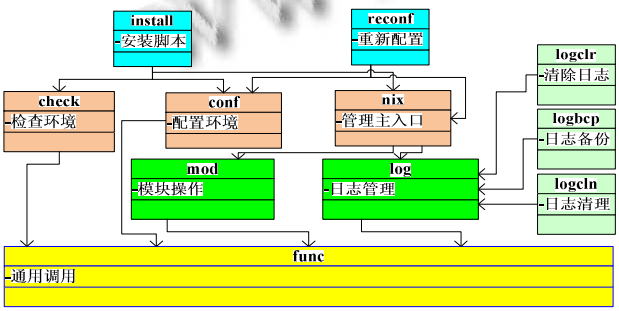


图 3 基础脚本之间引用和依赖关系

需要特别注意的是, 以上应用功能依赖于并由以下核心基础脚本构成, 其引用及依赖关系如图 4 所示, 包括: (1)nix, 部署和管理框架的主入口, 提供如表 1 所述的总体的系统界面, 使用了 Fa?ade 设计模式; (2)mod, 提供对组件的操作, 为 nix 正面的实际执行体, 执行对组件的如表 1 中说述组件操作; (3)log, 日志管理, 是 nix log 的实际执行体, 具体实现各类日志的显示和维护; (4)func, 通用调用包含文件, 作为其他几乎所有功能的基础函数库; (5)install、check、conf, 安装时的环境检查和配置, 实现框架安装前后工作, 提供重配置的基础; (7)logcln/logclr/logbc, 批量自动日志功能清理、清除和备份功能; (8)reconf, 系统组件的重配置功能, 执行时无需重装已注册的组件.

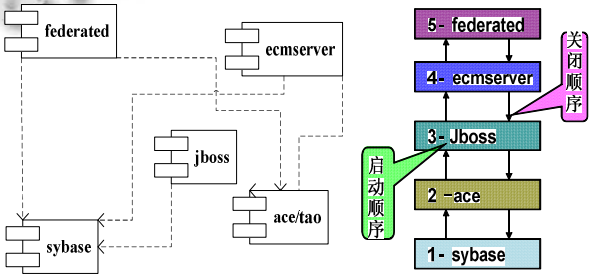


图 4 多组件的依赖性定义

### 2.4 框架系统的关键技术

#### 2.4.1 文件目录组织规范

为便于系统在进行各类安装、运行、启动、升级和日志记录时能自动定位到某些组件的位置, 需要制定一个文件目录的组织规范; 制定这个规范的目的, 一方面, 为具体操作时提供自动的文件和目录定位, 另外一方面, 为框架在可扩展方面提供良好的基础. 如表 1, 列出了框架及应用系统在安装(install)前后的目录比较, 可以发现, 通过对对目录分类, 可实现 mod1、mod2、mod3…的多个组件的支持, 为实现多组件的部署和管理等应用提供基础.

#### 2.4.2 多组件的依赖性

在实际的应用当中, 多个组件之间必定存在着相互的以来关系, 主要存在这几类:

(1)数据库组件必须在应用服务组件之前启动;

(2)基础处理组件必须在高级组件启动之前启动;

(3)本地应用组件必须在联网应用组件前启动;

(4)无直接关联的组件可自定义先后顺序.

以图 4 为例, 左侧为组件依赖关系, 其中虚线表示组件的依赖关系, 则可定义的组件的依赖性如右侧

图所示. 具体可在 func 中定义关系:

set NIX_MODS="sybase ace ecmserver jboss federated"

表 1　安装前后文件目录组织结构规范

```
1.安装前的目录结构:
./nixsetup---
|---./install //框架安装脚本
   |---(bin)   //可执行脚本存放目录
   |---(doc)   //帮助文档存放目录
   |---(mod)---    //符合基本格式的组件存放目录
      |---mod1     //实例组件 mod1 的程序启动脚本
         |---mod1.profile   //实例组件 mod1 的配置文件
         |---mod1.setup     //实例组件 mod1 个性化脚本
         |---mod1.tar.gz    //实例组件 mod1 的程序包
      |--(mod2)      //实例组件 mod2
         |---mod2    //类似 mod1 的目录结构
         ……
```

```
2.安装后的目录结构:
./some/dest/dir---
   |---(bak)    //数据库和程序备份存放目录
   |---(bin)    //可执行脚本存放目录
   |---(doc)    //帮助文档存放目录
   |---(etc)    //配置文件存放目录
   |---(log)    //程序输出日志存放目录
   |---(mod)--- //模块组件执行程序目录
      |--- (mod1)---
      |--- (mod2)---
      |--- (mod3)---
      |...
   |---(src)    //安装源和升级库
      ……
```

2.4.3 组件在线编译和自动升级

在本框架中, 组件具备较为完善的自动执行功能, 如自动备份(tar)、在线编译(make)和自动从本地或者从远程(通过 wget)进行升级. 如表 2 所述, 组件在安装时除拷贝和解压以外, 还使用%mod%.setup 定义了模块自己个性化设置, 如是否使用默认配置、是否在安装解压后执行在线重新编译(make clean all), 这点对于跨平台部署的编译特别有用.

## 3　框架的应用及定制

### 3.1 框架的实例化

本框架在实例化时, 如有需要, 可将 nix 命令改成产品名; 例如, 对于具体服务应用 ecm(实际为某电力公司通信网络的监控系统), 部署在 RedHat Linux 或者是 HP/Tru64 系统下, 其组件构成包括如下 5 个部分:

(1)Sybase: 数据库系统, 用以存储数据和物理模型; (2)ACE/TAO: 跨平台可适应通信和可移植 CORBA/C++编译、调试和运行环境; (3)JBOSS: 用以装业务应用在 J2EE 容器, 使用数据库连接池依赖并连接 Sybase 数据库; (4)ECMServer: 监控系统实时服务, 负责采集并处理告警或性能等实时数据, 运行时需要读取 Sybase 数据库中的数据内容; (5)Federated: 异地系统之间的联网程序, 组成低耦合的系统互联, 依赖于 ACE/TAO、JBOSS.

依据上述的依赖关系, 并结合 2.4.2 所述的依赖性定义方法, 可实现如图 4 所述启动顺序定义. 如图 5 所示为其部署安装完成后的系统视图, 各项结构和目录说明见图 5 中详细说明. 特别的, 对于 mod 目录, 按照文件目录组织规范, 可灵活增加相应的组件, 实现组件的可扩展部署.

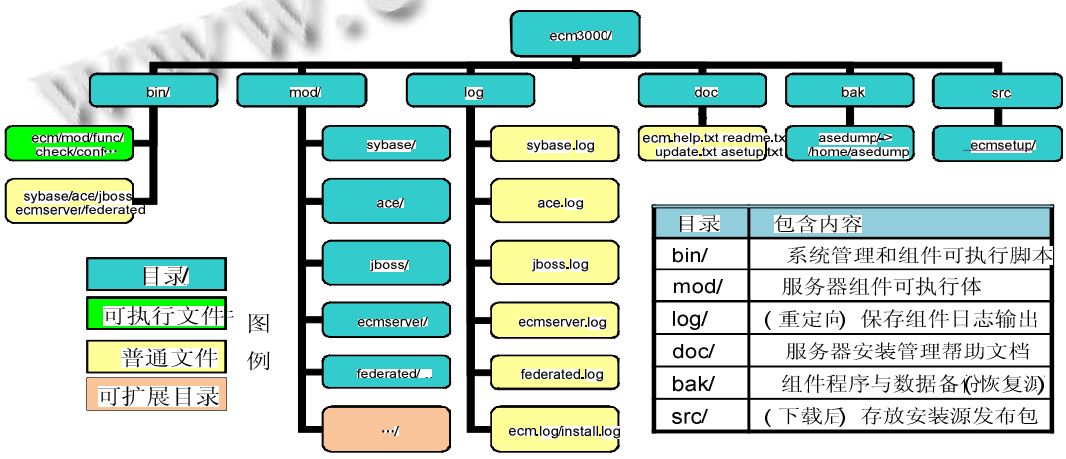


图 5　框架实例 ecm3000

### 3.2 应用实例的运行

ecm(nix 的实例化)的 CLI 命令界面视图如图 6, 介于篇幅, 暂只显示两条命令(show/status)的执行.

(1)ecm show 显示当前运行的实例的名称, 其进程是否在正常运行, 并且显示进程中的线程数(NLWP); (2)使用 ecm status 时, 则显示各个进程中的具体参数, 如 PID、CPU 使用率、内存、CPU 耗时和执行命令等, 如下图所示:

```
root@ecmserver:~# ecm show
================================================================================
[MANAGE] ECM3000_SERVER modset includes [sybase ace jboss ecmserver federated platform]
--------------------------------------------------------------------------------
[sybase    ] Sybase Database ASE 15 Linux  is running @ processes=4 | threads=4
[ace       ] ACE/TAO CORBA Naming Service  is running @ processes=2 | threads=2
[jboss     ] Java Application Server JBoss is running @ processes=1 | threads=42
[ecmserver] ECM RealTime Monitor Server    is running @ processes=1 | threads=228
[federated] ECM Federated Platform Server is running @ processes=9 | threads=634
[platform ] ECM Jiangsu Platform Server    is running @ processes=1 | threads=2
================================================================================
root@ecmserver:~# ecm status
================================================================================
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
--------------------------------------------------------------------------------
root      1894  0.0  0.1   5516  1172 ?        S    Oct15   0:00 RUN_ecm3000
root      1896  0.0  0.1   5516  1172 ?        S    Oct15   0:00 RUN_ecm3000_BS
root      1897  0.3  6.1  95432 62732 ?        Ssl  Oct15   4:48 dataserver
root      1898  0.0  5.4  57580 55660 ?        S    Oct15   0:00 backupserver
root      2108  0.0  0.6   9984  6776 ?        S    Oct15   0:00 Naming_Service
root      2129  0.0  0.6   9548  6380 ?        S    Oct15   0:00 Naming_Service
root      2308  0.3  8.5 532900 87488 ?        Sl   Oct15   4:21 java
root      2506  0.0  1.9 2391092 20264 ?       Sl   Oct15   0:00 .ecmserver
root      2687  0.0  1.1  57692 11836 ?        Sl   Oct15   0:00 .Federated_CorbaServer
root      2689  0.0  0.8  54112  8160 ?        Sl   Oct15   0:00 .CorbaServer_CenterTem
root      2707  0.0  0.9  76404  9212 ?        Sl   Oct15   0:00 .Federated_ServerProxy
root      2708  0.0  0.7  21980  7496 ?        Sl   Oct15   0:00 .SecurityServer
root      2726  0.0  0.7  11624  7472 ?        S    Oct15   0:00 .Federated_Shared_EC
root      2747  0.0  1.0 3119752 10272 ?       Sl   Oct15   0:00 .Federated_State_Proce
root      2773  0.0  1.2 3120832 12524 ?       Sl   Oct15   0:00 .Federated_Alarm_Suppl
root      3093  0.0  0.8  43824  8164 ?        Sl   Oct15   0:00 .Federated_Alarm_Consu
root      3413  0.0  0.7  21964  7484 ?        Sl   Oct15   0:00 .SendAlarmSvr
root      3566  0.0  1.8  68568 18816 ?        Sl   Oct15   0:03 .SecurityServer
================================================================================
```

图 6 框架实例运行 CLI 界面输出

## 4 结语

通过基于 BASH 脚本的部署和管理框架, 结合具有良好的可操作性、可扩展性和兼容性, 为 Unix 下多组件的安装、运行、日志、监视、备份、自动升级、清理、帮助提供了良好的应用基础. 但是 Unix 的应用时各种各样, 针对具体的应用, 应当对本框架实例化并进行适当的改造和扩展, 才能适应具体的应用需要, 这一点需要特别注意的.

## 参考文献


1 石庆冬.例谈 Bash 与 Tcsh 的主要区别.电脑知识与技术,2008,(33).

2 石庆冬.浅谈 Bash 与 Bourne shell 的主要区别.大众科技,2008,(11):32–33.

3 高尚,史兴娟,丰立东,等. 一种基于 Bash shell 脚本板件功能测试方法.中国专利数据库,2010,(10),CN201010120390.8.

4 张勇.基于类 Unix 系统中 shell 脚本的中文文本字频分析.才智,2011,32.

5 蔡卫东.正确调用自动化系统的 shell 脚本.空中交通管理,2008(12).

6 闫格,郑艺峰.基于 Shell 脚本的 IBM Cell/B.E.模拟环境的自动安装与测试.漳州师范学院学报(自然科学版),2009,(3).

7 张根宝,胡杰.Linux 集群环境下监控 Web 服务器的 Shell 脚本设计.化工自动化及仪表,2010,(10).

8 江松波,倪子伟.浅谈自底向上的 Shell 脚本编程及效率优化.计算机与现代化,2011,(2).

9 韩璐.在 UNIX 系统下用 shell 编程实现对文件的操作.中国科技信息,2006,(13).


（上接第 39 页）


1:1997. Copyright 1997 The VRMLConsortium Incorporated.

3 Cult3D Homepage. http://www.cult3D.com.

4 冯凤娟.永丰大厦虚拟漫游数字平台的设计与实现.北京:北京交通大学,2010.

5 韩万江,姜立新.系统工程与软件工程.计算机应用,2010,(S1):212–214.

6 Schach SR.软件工程面向对象和传统的方法.邓迎春,等译.北京:机械工业出版社,2009.

7 袁丽娜,赵贵斌.碰撞检测技术及在桥梁视景仿真中的应用.工程图学学报,2010,(3):21–26.

8 李远鑫,蒋海鸥,徐亦飞,徐芝琦.基于 Web3D 的交互式虚拟社区.计算机工程,2011,37(11):287–290.

9 黄健柏,邹峥嵘,朱学红.虚拟校园及其在校园规划管理中的应用.教育信息化,2002(6):7–8.