BSD rc.d脚本□程□□

摘要

初学者可能会□□，□以通□正式的文□， 基于 BSD 的 rc.d 框架，□写一些□□任□的 rc.d 脚本。 本文中，我□采用了一些□□性不断□加的典型案例， 来展示□合□个案例的 rc.d 特性， 并探□其中的工作原理。 □□的□□□大家□一□研究□□有效的 rc.d □用程序提供了一些参考点。

目□

1. □介

□史上 BSD 曾有□一个□一的□□脚本， /etc/rc。 □脚本在系□□□的□候被 init(8) 程序所引□，并□行所有多用□操作所需求的用□□任□： □□并挂□文件系□，□置网□，□□守□□程，等等。 在□个系□中□□的任□清□也并不相同； 管理□需要根据需求自定□□□的任□清□。在一些特殊的情况中， □不得不去修改 /etc/rc 文件，一些真正的□客□此不疲。

□一脚本□□方法的真正□□是它没有提供□从 /etc/rc □□的□个□件的控制。 拿一个例子来□□，/etc/rc 不能□重新□□某个□独的守□□程。 系□管理□不得不手□□出守□□程，并□掉它， 等待它真正退出后，再通□□□ /etc/rc 得到□守□□程的□□，最□□入全部命令来再次□□守□□程。 如果重新□□的服□包括不止一个守□□程或需要更多□作的□，□任□将□得更加困□以及容易出□。□而言之， □一脚本在□□我□□□的目的上是不成功的： □系□管理□的生活更□松。

再后来，□了将最重要的一些子系□独立出来， 便□□将部分的内容从 /etc/rc 分□出来了。 最广□人知的例子就是用来□□□网的 /etc/netstart 文件。它容□从□用□模式□□网□， 但由于它的部分代□需要和一些与□网完全无□的□作交互， 所以它并没有完美地□合到自□□的□程中。那便是□何 /etc/netstart 被演□成 /etc/rc.network 的原因了。 后者不再是一个普通的脚本；它包括了□大的，由 /etc/rc 在不同的系□□□□□中□用的凌乱的 sh(1) 函数。然而，当□□任□□得多□化以及久□更改， "□模□化" 方法□得比曾□的整体 /etc/rc 更□慢□事。

由于没有一个干□和易于□□的框架， □□脚本不得不全力更改以□足□速□□中基于 BSD 的操作系□的需求。它逐□□得明朗并□□□多必要的□□最□□成一个具有□密性和□展性的 rc 系□。BSD rc.d 就□□□生了。Luke Mewburn 和 NetBSD 社区是公□的 rc.d 之父。再之后它被引入到了 FreeBSD 中。 它的名字引用□系

□□独的服□脚本的位置，也就是 /etc/rc.d 下面的那些脚本。 之后我□将学□到更多的 rc.d 系□的□件并看看□个脚本是如何被□用的。

BSD rc.d 背后的基本理念是 良好 的模□化和代□重用性。 良好 的模□化意味着□个基本 "服□" 就象系□守□□程或原始□□任□那□， 通□属于它□的可□□□服□的 sh(1) 脚本，来停止服□， 重□服□，□□服□的状□。具体□作由脚本的命令行参数所决定。 /etc/rc 脚本仍然掌管着系□的□□， 但□在它□□是使用 `start` 参数来一个个□用那些小的脚本。 □便于用 `stop` 来□□行中的同□的脚本很好地□行停止任□， □是被 /etc/rc.shutdown 脚本所完成的。看，□是多□好地体□了 Unix 的哲学： □有一□小的□用的工具，□个工具尽可能好地完成自己的任□。 代□重用 意味着所有的通用操作由 /etc/rc.subr 中的一些 sh(1) 函数所□□。 □在一个典型的脚本只需要□□几行的 sh(1) 代□。最□，rcorder(8) 成□了 rc.d 框架中重要的一部分， 它用来□助 /etc/rc □理小脚本之□的依□□系并有次序地□行它□。它同□□助 /etc/rc.shutdown 做□似的事情， 因□正□的□□次序是相□于□□的次序的。

BSD rc.d 的□□在 Luke Mewburn 的原文 中有□□， 以及 rc.d □件也被充分□□地□□在各自的 □机手册中。然而， 它可能没能清晰展□□一个 rc.d 新手，如何将无数的□和片□行□□来□具体的任□□建一个好□格的脚本。 因此本文将□着以不同的方式来□述 rc.d。 它将展示在某些典型情况中□□使用□些特性，并□述了□何如此。 注意□并不是一篇 how-to 文□，我□的目的不是□出□成的配方， 而是在展示一些□□的□入 rc.d 的□□的□路。 本文也不是相□□机手册的替代品。 □□本文□□得同□参考□机手册以□取更完整正□的文□。

理解本文需要一些先决条件。首先，□需要熟悉 sh(1) 脚本□程□言以掌握 rc.d， □有，□需要知道系□是如何□行用□□的□□和停止任□，□些在 rc(8) 中都有□明。

本文□注的是 rc.d 的 FreeBSD 分支。 不□，它可能□ NetBSD 的□□者也同□有用，因□ BSD rc.d 的□个分支不只是共享了同□的□□， □保留了□脚本□写者都可□的□似□点。

2. 任□描述

在□始打□ `$EDITOR`（□□器） 之前□行小小的思考不是坏事。□了□一个系□服□写一个 "听□的"rc.d 脚本， 我□首先□□能回答以下□□：

- □服□是必□性的□是可□性的？
- 脚本将□□个程序服□，如一个守□□程，□是□行更□□的□作？
- 我□的服□依□□些服□？反□来□些服□依□我□的服□？

从下面的例子中我□将看到，□什□□知道□些□□的答案是很重要的。

3. 虚□的脚本

下面的脚本是用来在□次系□□□□□出一个信息：

```
#!/bin/sh ①

. /etc/rc.subr ②

name="dummy" ③
start_cmd="${name}_start" ④
stop_cmd=":" ⑤

dummy_start() ⑥
{
    echo "Nothing started."
}

load_rc_config $name ⑦
run_rc_command "$1" ⑧
```

需要注意的是：

□ 一个解□性的脚本□□以一行魔幻的 "shebang" 行□□。 □行指定了脚本的解析程序。由于 shebang 行的作用， 假如再有可□行位的□置， 脚本就能象一个二□制程序一□被精□地□用□行。 （□参考 chmod(1)。） 例如， 一个系□管理□可以从命令行手□□行我□的脚本：

```
# /etc/rc.d/dummy start
```

□了使 rc.d 框架正□地管理脚本， 它的脚本需要用 sh(1) □言□写。 如果□的某个服□或 port 套件使用了二□制控制程序或是用其它□言□写的例程， □将其□件安装到 /usr/sbin（相□于系□） 或 /usr/local/sbin（相□于ports）， 然后从合□的 rc.d 目□的某个 sh(1) 脚本□用它。

如果□想知道□什□ rc.d 脚本必□用 sh(1) □言□写的□□，先看下 /etc/rc 是如何依□ `run_rc_script` □用它□， 然后再去学□ /etc/rc.subr 下 `run_rc_script` 的相□□□。

□ 在 /etc/rc.subr 下， 有□多定□□的 sh(1) 函数可供□个 rc.d 脚本使用。□些函数在 rc.subr(8) 中都有□明。尽管理□上可以完全不使用 rc.subr(8) 来□写一个 rc.d 脚本，但它的函数已□□明了它真的很方便，并且能使任□更加的□□。所以所有人在□写 rc.d 脚本□都会求助于 rc.subr(8) 也不足□奇了。当然我□也不例外。

一个 rc.d 脚本在其□用 rc.subr(8) 函数之前必□先 "source"/etc/rc.subr（用 "`.`"将其包含□去）， 而使 sh(1) 程序有机会来□悉那些函数。 首□□格是在脚本的最□始 source /etc/rc.subr 文件。

某些有用的与□网有□的函数由□一个被包含□来的文件提供， /etc/network.subr 文件。

□ □制的□量 `name` 指定我□脚本的名字。 □是 rc.subr(8) 所□□的。也就是， □个 rc.d 脚本在□用 rc.subr(8) 的函数之前必□□置 `name` □量。

□在是□候来□我□的脚本一次性□□一个独一无二的名字了。 在□写□个脚本的□我□将在□多地方用到它。在□始之前， 我□来□脚本文件也取个相同的名字。

当前的 rc.d 脚本□格是把□放在双引号中来□□量□□。 □□住□只是个□格□□，可能并不□是□□。 □可以在只是□□的并不包括 sh(1) 元字符的□句中放心地省略掉引号， 而在某些情况下□将需要使用□引号以防止 sh(1) □任何的□量的解□。 程序□是可以□巧地由□格□例□悉其□法以及使用的。

□ rc.subr(8) 背后主要的□思是 rc.d 脚本提供□理程序，或者方法，来□ rc.subr(8) □用。特□是，`start`，`stop`，以及其它的 rc.d 脚本参数都是□□被□理的。方法是存□在一个以 `argument_cmd` 形式命名的□量中的 sh(1) 表□式，□ *argument* □□着脚本命令行中所特□指定的参数。我□□后将看到 rc.subr(8) 是如何□□准参数提供默□方法的。

□了□ rc.d 中的代□更加□一， 常□的是在任何□合的地方都使用 `${name}` 形式。□□一来，可以□松地将一些代□从一个脚本拷□到□一个中使用。

□ 我□□□□ rc.subr(8) □□准参数提供了默□的方法。 因此，如果希望它什□都不做的□，我□必□使用无操作的 sh(1) 表□式来改写□准的方法。

□ 比□□□的方法主体可以用函数来□□。 在能□保□函数名有意□的情况下，□是个很不□的想法。

□烈推□□我□脚本中所定□的所有函数名都添加□似 `${name}` □□的前□，以使它□永□不会和 rc.subr(8) 或其它公用包含文件中的函数冲突。

□ □是在□求 rc.subr(8) □入 rc.conf(5) □量。 尽管我□□个脚本中使用的□量并没有被其它地方使用，但由于 rc.subr(8) 自身所控制着的 rc.conf(5) □量存在的原因，仍然推□脚本去装□ rc.conf(5)。

□ 通常□是 rc.d 脚本的最后一个命令。 它□用 rc.subr(8) 体系使用我□脚本所提供的□量和方法来□行相□的□求□作。

4. 可配置的虚□脚本

□在我□来□我□的虚□脚本□加一些控制参数□。正如□所知， rc.d 脚本是由 rc.conf(5) 所控制的。 幸□的是，rc.subr(8) □藏了所有□□化的□西。 下面□个脚本使用 rc.conf(5) 通□ rc.subr(8) 来□看它是否在第一个地方被□用，并□取一条信息在□□□□示。 事□上□□个任□是相互独立的。一方面，rc.d 脚本要能□支持□□和禁用它的服□。□一方面， rc.d 脚本必□能具□配置信息□量。 我□将通□下面同一脚本来演示□□方面的内容：

```
#!/bin/sh

. /etc/rc.subr

name=dummy
rcvar=dummy_enable ①

start_cmd="${name}_start"
stop_cmd=":"

load_rc_config $name ②
eval "${rcvar}=\${${rcvar}:-'NO'}" ③
dummy_msg=${dummy_msg:-"Nothing started."} ④

dummy_start()
{
    echo "$dummy_msg" ⑤
}

run_rc_command "$1"
```

在□个□例中改□了什□？

□ □量 rcvar 指定了 ON/OFF □□□量的名字。

□ □在 load_rc_config 在任何 rc.conf(5) □量被□□之前就在脚本中被□先□用。

□□ rc.d 脚本□，切□ sh(1) 会把函数延□到其被□用□才□其中的表□式□行求□□算。因此尽可能□地在 run_rc_command 之前□用 load_rc_config，以及仍然□□从方法函数□出到 run_rc_command 的 rc.conf(5) □量并不是一个□□。□是因□方法函数将在 load_rc_config 之后，被□用的 run_rc_command □用。

□ 如果自身□置了 rcvar，但指示□□□量却未被□置，那□ run_rc_command 将□出一个警告。如果□的 rc.d 脚本是□基本系□所用的，□□当在 /etc/defaults/rc.conf 中□□□□量添加一个默□的□置并将其□注在 rc.conf(5) 中。否□的□□的脚本□□□□□□量提供一个默□□置。□例中演示了一个可移植接近于后者情况的案例。

□可以通□将□□□量□置□ ON 来使 rc.subr(8) 有效，使用 one 或 force □脚本的参数加前□，如 onestart 或 forcestop □□，会忽略其当前的□置。切□ force 在我□下面要提到的情况下有□外的危□后果，那就是当用 one 改写了 ON/OFF □□□量。例如，假定 dummy_enable 是 OFF 的，而下面的命令将忽略系□□置而□行□行 start 方法：

```
# /etc/rc.d/dummy onestart
```

□ □在□□□□示的信息不再是硬□□在脚本中的了。它是由一个命名□ dummy_msg 的 rc.conf(5) □量所指定的。□就是 rc.conf(5) □量如何来控制 rc.d 脚本的一个小例子。

我□的脚本所独占使用的所有 rc.conf(5) □量名， 都必□具有同□的前□：${name}。例如：dummy_mode， dummy_state_file，等等。

当可以内部使用一个□短的名字□，如 msg □□，□我□的脚本所引□的全部的共用名添加唯一的前□ ${name}，能□避免我□与 rc.subr(8) 命名空□冲突的可能。

只要一个 rc.conf(5) □量与其内部等同□是相同的， 我□就能□使用一个更加兼容的表□式来□置默□□：

```
: ${dummy_msg:="Nothing started."}
```

尽管目前的□格是使用了更□□的形式。

通常，基本系□的 rc.d 脚本不需要□它□的 rc.conf(5) □量提供默□□， 因□默□□□□是在 /etc/defaults/rc.conf □置□了。但□一方面，□ ports 所用的 rc.d 脚本□提供如□例所示的默□□置。

□ □里我□使用 dummy_msg 来□□地控制我□的脚本，即，□一个□量信息。

5. □□并停止□□守□□程

我□早先□□ rc.subr(8) 是能□提供默□方法的。 □然，□些默□方法并不是太通用的。 它□都是□用于大多数情况下来□□和停止一个□□的守□□程况。 我□来假□□在需要□一个叫做 mumbled 的守□□程□写一个 rc.d 脚本， 在□里：

```
#!/bin/sh

. /etc/rc.subr

name=mumbled
rcvar=mumbled_enable

command="/usr/sbin/${name}" ①

load_rc_config $name
run_rc_command "$1"
```

感到很□□□，不是□？我□来□□下我□□个小脚本。 只需要注意下面的□些新知□点：

□ □个 command □量□于 rc.subr(8) 来□是有意□的。当它被□置□， rc.subr(8) 将根据提供□□守□□程的情形而生效。 特□是，将□□些参数提供默□的方法： start， stop， restart， poll， 以及 status。

□守□□程将会由□行中的 $command 配合由 $mumbled_flags 所指定的命令行□□来□□。 因此，□默□的 start 方法来□， 所有的□入数据在我□脚本□量集合中都可用。与 start 不同的是， 其他方法可能需要与□程□□相□的□外信息。□个例子， stop 必□知道□程的 PID 号来□□□程。 在目前的情况中，rc.subr(8) 将□描全部□程的清□， □□一个名字等同于 $procname 的□程。 后者是□一个□ rc.subr(8) 有意□的□量， 并且默□它的□跟

command 一□。 □而言之，当我□□ command □置□后， procname □□上也□置了同□的□。 □□□我□的脚本来□死守□□程并□□它是否正在第一个位置□行。

某些程序□□上是可□行的脚本。 系□□□脚本的解□器以□□脚本名□命令行参数的形式来□行脚本。 然后被映射到□程列表中，□会使 rc.subr(8) 迷惑。因此，当 $command 是一个脚本的□，□□□□外地□置 command_interpreter 来□ rc.subr(8) 知□□程的□□名字。

□□个 rc.d 脚本而言， 有一个可□的 rc.conf(5) □量□ command 指示其□先□。它的名字是下面□□的形式：${name}_program， name 是我□ 之前 □□□的必□性□量。如，在□个案例中它□□命名□ emumbled_program。□其□是 rc.subr(8) 分配 ${name}_program 来改写 command 的。

当然，即使 command 未被□置， sh(1) 也将允□□从 rc.conf(5) 或自身来□置 ${name}_program。在那□情况下， ${name}_program 的特定属性□失了， 并且它成□了一个能供□的脚本用于其自身目的的普通□量。 然而，□独使用 ${name}_program 是并不是我□所寄望的，因□同□使用它和 command 已成□了 rc.d 脚本□程的一个□用的□定。

□于默□方法的更□□的信息，□参考 rc.subr(8)。

6. □□并停止高□守□□程

我□来□之前的 "骨架" 脚本加点 "血肉"，并□它更□□更富有特性□。 默□的方法已能□□我□做很好的工作了，但是我□可能会需要它□一些方面的□整。 □在我□将学□如何□整默□方法来符合我□的需要。

```
#!/bin/sh

. /etc/rc.subr

name=mumbled
rcvar=mumbled_enable

command="/usr/sbin/${name}"
command_args="mock arguments > /dev/null 2>&1" ①

pidfile="/var/run/${name}.pid" ②

required_files="/etc/${name}.conf /usr/shared/misc/${name}.rules" ③

sig_reload="USR1" ④

start_precmd="${name}_prestart" ⑤
stop_postcmd="echo Bye-bye" ⑥

extra_commands="reload plugh xyzzy" ⑦

plugh_cmd="mumbled_plugh" ⑧
xyzzy_cmd="echo 'Nothing happens.'"

mumbled_prestart()
```

```
{
    if checkyesno mumbled_smart; then ⑨
        rc_flags="-o smart ${rc_flags}" ⑩
    fi
    case "$mumbled_mode" in
    foo)
        rc_flags="-frotz ${rc_flags}"
        ;;
    bar)
        rc_flags="-baz ${rc_flags}"
        ;;
    *)
        warn "Invalid value for mumbled_mode" ⑪
        return 1 ⑫
        ;;
    esac
    run_rc_command xyzzy ⑬
    return 0
}

mumbled_plugh() ⑭
{
    echo 'A hollow voice says "plugh".'
}

load_rc_config $name
run_rc_command "$1"
```

□ 附加□ $command 的参数在 command_args 中□行□□。它□在 $mumbled_flags 之后将被添加到命令行。 其□□的□行便是此后最□的命令行□□□ eval □算，□入和□出以及重定向都可以在 command_args 中指定。

永□不要 在 command_args 包含破折号□□， □似 -X 或 --foo □□的。command_args 的内容将出□在最□命令行的末尾，因此它□可能是□接在 ${name}_flags 中所列出的参数后面； 但大多的命令将不能□□出普通参数后的破折号□□。 更好的□□附加□ $command 的□□的方式是添加它□到 ${name}_flags 的起始□。□一□方法是像后文所示的那□来修改 rc_flags。

□ 一个得体的守□□程会□建一个 *pidfile* □程文件， 以使其□程能□更容易更可□地被□到。如果□置了 pidfile □量，告□ rc.subr(8) □里能□到供其默□方法所使用的 pidfile □程文件。

事□上，rc.subr(8) 在□□一个守□□程前□会使用 pidfile □程文件来□看它是否已□在□行。使用了 faststart 参数可以跳□□个□□□□。

□ 如果守□□程只有在□定的文件存在的情况下才可以□行， 那就将它□列到 required_files 中，而 rc.subr(8) 将在□□守□□程之前□□那些文件是否存在。 □有相□的分□用来□□目□和□境□量的 required_dirs 和 required_vars 可供使用。它□都在 rc.subr(8) 中有□□的□明。

来自 rc.subr(8) 的默□方法，通□使用 forcestart 作□脚本的参数， 可以□制性地跳□□先需要的□□。

□ 我□可以在守□□程有□常的□候，自定□□送□守□□程的信号。 特□是，sig_reload 指定了使守□□程重新装□其配置的信号；默□情况也就是 SIGHUP 信号。 □一个信号是□送□守□□程以停止□□程；默□情况下是 SIGTERM 信号，但□是可以通□□置 sig_stop 来□行□当更改的。

信号名称□当以不包含 SIG 前□的形式指定□ rc.subr(8)，就如□例中所示的那□。 FreeBSD 版本的 kill(1) 程序能□□□出 SIG 前□，不□其它系□版本的就不一定了。

□□ 在默□的方法前面或后面□行附加任□是很容易的。 □于我□脚本所支持的□条命令参数而言，我□可以定□ argument_precmd 和 argument_postcmd 来完成。□些 sh(1) 命令分□在它□各自的方法前后被□用， □然，从它□各自的名字就能看出来。

如果我□需要的□，用自定□的 argument_cmd 改写默□的方法，并不妨碍我□仍然使用 argument_precmd 和 argument_postcmd。 特□是，前者便于□□自定□的方法， 以及□行自身命令之前所遇到更□密的条件。于是，将 argument_precmd 和 argument_cmd 一起使用，使我□合理地将□□从□作中独立了出来。

□忘了□可以将任意的有效的 sh(1) 表□式□入到方法和□定□的 pre- 与 post-commands 命令中。 在大部分情况下，□用函数使□□任□有好的□格， 但千万不要□□格限制了□□其幕后到底是□□回事的思考。

□ 如果我□□意□□一些自定□参数， □些参数也可被□作□我□脚本的 命令，我□需要在 extra_commands 中将它□列出并提供方法以□理它□。

reload 是个特□的命令。一方面， 它有一个在 rc.subr(8) 中□置的方法。□一方面， reload 命令默□是不被提供的。 理由是并非所有的守□□程都使用同□的重□方法， 并且有些守□□程根本没有任何□西可重□的。所以□而易□， 我□需要去□□都提供了□些的内建功能。我□可以通□ extra_commands 来□□做。

我□从 reload 的默□方法得到了什□□？ 守□□程常常在收到一个信号后重新□入它□的配置 - 一般来□，也就是 SIGHUP 信号。因此 rc.subr(8) □□□送一个信号□守□□程来重□它。□信号一般□□□ SIGHUP， 但是如果必要的□可以通□ sig_reload □量来自定□它。

□□ 我□的脚本提供了□个非□准的命令， plugh 和 xyzzy。 我□看到它□在 extra_commands 中被列出来了，并且□在是□候□它□提供方法了。xyzzy 的方法是内□的而 plugh 的是以 mumbled_plugh 形式完成的函数。

非□准命令在□□或停止的□候不被□用。 通常它□是□了系□管理□的方便。它□□能被其它的子系□所使用，例如， devd(8)，前提是 devd.conf(5) 中已□指定了。

全部可用命令的列表，当脚本不加参数地□用□，在 rc.subr(8) 打印出的使用方法中能□□到。例如，□就是供学□的脚本用法的内容：

```
# /etc/rc.d/mumbled
Usage: /etc/rc.d/mumbled [fast|force|one]
(start|stop|restart|rcvar|reload|plugh|xyzzy|status|poll)
```

□ 如果脚本需要的□，它可以□用自己的□准或非□准的命令。 □可能看起来有点像函数的□用，但我□知道，命令和 shell 函数并非一直都是同□的□西。□个例子，xyzzy 在□里不是以函数来□□的。□外，□有□□被有序□用的 pre-command □置命令和 post-command 后置命令。 所以脚本□行自己命令的合

□方式就是利用 rc.subr(8)， 就像□例中展示的那□。

□ rc.subr(8) 提供了一个方便的函数叫做 checkyesno。 它以一个□量名作□参数并返回一个□零的退出□，当且□当□□量□置□ YES，或 TRUE，或 ON，或 1，区分大小写；否□返回一个非零的退出□。 在第二□情况中，函数□□□量的□置□ NO， FALSE，OFF，或 0，区分大小写； 如果□量包含□的内容的□它打印一条警告信息，例如，□□。

切□□ sh(1) 而言零□意味着真而非零□意味着假。

checkyesno 函数使用一个 □量名。不要□大含□将□量的 □ □□□它； 否□它不会如□□期那□的工作。

下面是 checkyesno 的合理使用□□：

```
if checkyesno mumbled_enable; then
        foo
fi
```

相反地，以下面的方式□用 checkyesno 是不会工作的 — 至少是不会如□□期的那□：

```
if checkyesno "${mumbled_enable}"; then
        foo
fi
```

□ 我□可以通□修改 $start_precmd 中的 rc_flags 来影□□□到 $command 的□□。

□ 某□情况下我□可能需要□出一条重要的信息，那□的□ syslog 可以很好地□□日志。 □可以使用下列 rc.subr(8) 函数来□松完成： debug， info， warn，以及 err。 后者以指定的代□□退出脚本。

□ 方法的退出□和它□的 pre-commands □命令不只是默□被忽略掉。如果 argument_precmd 返回了一个非零退出□，主方法将不会被□行。依次地是， argument_postcmd 将不会被□用，除非主方法返回的是一个□零的退出□。

然而，当□一个参数使用 force 前□的□候，如 forcestart，rc.subr(8) 会听从命令行指示而忽略那些退出□最后仍然□用所有的命令。

7. □接脚本到 rc.d 框架

当□写好了一个脚本，它需要被整合到 rc.d 中去。 一个重要的□□就是安装脚本到 /etc/rc.d （□基本系□而言）或 /usr/local/etc/rc.d （□ports而言）中去。在 bsd.prog.mk 和 bsd.port.mk 中都□此提供了方便的接口， 通常□不必担心□当的所有□限和模式。系□脚本□当是通□可以在 src/etc/rc.d □到的 Makefile 安装的。Port 脚本可以像 Porter's Handbook 中描述那□通□使用 USE_RC_SUBR 来被安装。

不□，我□□□先考□到我□脚本在系□□□序中的位置。 我□的脚本所□理的服□可能依□于其它的服□。□个例子，没有网□接口和路由□□的□用□行的□，一个网□守□□程是不起作用的。 即使一个服□看似什□都不需要，在基本文件系□□□挂□完□之前也很□□。

之前我□曾提到□ rcorder(8)。□在是□候来密切地□注下它了。 □□地□，rcorder(8) □理一□文件，□□它□的内容， 并从文件集合打印一个文件列表的依□□序到 stdout □准□出。□点是用于保持文件内部的依□信息，而□个文件只能□明自己的依□。一个文件可以指定如下信息：

- 它 提供 的 "条件" 的名字（意味着我□服□的名字）；
- 它 需求 的 "条件" 的名字；
- □□ 先 □行的文件的 "条件"的名字；
- 能用于从全部文件集合中□□一个子集的□外 □□字（ rcorder(8) 可通□□□而被指定来包括或省去由特殊□□字所列出的文件。）

并不奇怪的是，rcorder(8) 只能□理接近 sh(1) □法的文本文件。rcorder(8) 所解□的特殊行看起来□似 sh(1) 的注□。□□特殊文本行的□法相当□格地□化了其□理。 □□□ rcorder(8) 以□取更□□的信息。

除使用 rcorder(8) 的特殊行以外， 脚本可以□持将其依□的其它服□□制性□□。当其它服□是可□的， 并因系□管理□□□地在 rc.conf(5) 中禁用掉□服□而使其不能自行□□□，会需要□一点。

将□些□□在心，我□来考□下□□□合了依□信息□□的守□□程脚本：

```
#!/bin/sh

# PROVIDE: mumbled oldmumble ①
# REQUIRE: DAEMON cleanvar frotz②
# BEFORE:  LOGIN ③
# KEYWORD: nojail shutdown ④

. /etc/rc.subr

name=mumbled
rcvar=mumbled_enable

command="/usr/sbin/${name}"
start_precmd="${name}_prestart"

mumbled_prestart()
{
    if ! checkyesno frotz_enable && \
        ! /etc/rc.d/frotz forcestatus 1>/dev/null 2>&1; then
        force_depend frotz || return 1 ⑤
    fi
    return 0
}

load_rc_config $name
run_rc_command "$1"
```

跟前面一□，做如下□□分析：

□ □行声明了我□脚本所提供的 "条件" 的名字。 □在其它脚本可以用那些名字来□明我□脚本的依□。

通常脚本指定一个□独的已提供的条件。然而，　　并没有什□妨碍我□从列出的那些条件中指定，例如，　□了兼容性的目的。

在其它情况，主要的名称，或者□唯一的，　PROVIDE: 条件□□与 ${name} 相同。

□□　　因此我□的脚本指示了其依□于□的脚本所提供的　　"条件"。根据□些行的信息，脚本□示　　rcorder(8) 以将其放在一个或多个提供 DAEMON 和 cleanvar 的脚本后面，但在提供 LOGIN 的脚本前面。

BEFORE:　　□一行不可以在其它脚本不完整的依□□系列表中□用。　　□合使用　　BEFORE: 的情况是当其它脚本不□心我□的脚本，　但是我□的脚本如果在□一个之前□行的□能□更好地□行任□。　一个典型的□例是网□接口和防火□：　□然接口不依□防火□来完成自己的工作，但是系□安全将因一切网□流量之前□□的防火□而受益。

除了条件相□□的□个□独服□，脚本使用元条件和它□的　　"占位符"　　来保□某个操作□在其它之前被□行。　□些是由 UPPERCASE 大写名字所表示的。它□的列表和用法可以在 rc(8) 中□到。

切□将一个服□名称放□　　REQUIRE:　　行不能保□□□的服□会在我□的脚本□□的□候□行。所需求的服□可能会□□失□或在 rc.conf(5) 中被禁掉了。　□然，rcorder(8) 是无法追踪□些□□的，并且　rc(8)　也不会去追踪。所以，　脚本□□的□用程序□当能□□付任何所需求的服□的不可用情况。 某些情况下，我□可以用 下面 所□□的方式来□助脚本。

□ 如我□从上述文字所□起的，rcorder(8) □□字可以用来□□或省略某些脚本。即任何 rcorder(8) 用□可以通□指定 -k 和 -s □□来分□指定 "保留清□（keep list）" 和 "跳□清□（skip list）"。 从全部文件到按依□□系排列的清□，rcorder(8)　将只是挑出保留清□（除非是空的）　中那些□□□字的以及从跳□清□中挑出不□□□字的文件。

在 FreeBSD 中，rcorder(8) 被 /etc/rc 和 /etc/rc.shutdown 所使用。 □□个脚本定□了 FreeBSD 中 rc.d □□字以及它□的意□的□准列表如下：

□ 以 force_depend 起始的行□被用于更□慎的情况。通常，用于修正相互□□的 rc.d 脚本分□□□的配置文件□会更加□妥。

如果□仍不能完成不含 force_depend 的脚本，□例提供了一个如何有条件地□用它的□□用法。在□例中，我□的 mumbled 守□□程需求□一个以高□方式□□的□程，　frotz。但　frotz　也是可□的；　而且　rcorder(8) □□些信息是一无所知的。幸□的是，　我□的脚本已□□到全部的 rc.conf(5) □量。如果 frotz_enable □真，我□希望的最好□果是依□ rc.d 已□□□了 frotz。 否□我□□制□□ frotz 的状□。最□，　如果 frotz 依□的服□没有□到或□行的□，　我□将□制其□行。□□　force_depend　将□出一条警告信息，因□它只□□在□□到配置信息□失的情况下被□用。

8. □予 rc.d 脚本更多的□活性

当□行□□或停止的□用□，rc.d　脚本□□作用于其所□□的整个子系□。例如，　/etc/rc.d/netif　□□□□或停止 rc.conf(5)　中所描述的全部网□接口。□个任□都唯一地听从一个如　start　或　stop　□□的□独命令参数的指示。在□□和停止之□的□□，　rc.d　脚本□助管理□控制□行中的系□，　并其在需要的□候它将□生更多的□活性和精□性。□个例子，　管理□可能想在　rc.conf(5)　中添加一个新网□接口的配置信息，然后在不妨碍其它已存在接口的情况下将其□□。　在下次管理□可能需要□□一个□独的网□接口。在魔幻的命令行中，□□的 rc.d 脚本□用一个□外的参数，网□接口名即可。

幸□的是，rc.subr(8) 允□□□任意多（取决于系□限制）的参数□脚本的方法。 由于□个原因，脚本自身的改□可以□是微乎其微。

rc.subr(8) 如何□□到附加的命令行参数□？直接□取□？ 并非是无所不用其□的。首先，sh(1) 函数没有□□到□用者的定位参数，而 rc.subr(8) 只是□些函数的容器。其次，rc.d 指令的一个好的□格是由主函数来决定将□些参数□□□它的方法。

所以 rc.subr(8) 提供了如下的方法： run_rc_command □□其所有参数但将第一个参数逐字□□到各自的方法。首先， □出以方法自身□名字的参数：start， stop，等等。□会被 run_rc_command 移出，□□命令行中原本 $2 的内容将作□ $1 来提供□方法，等等。

□了□明□点，我□来修改原来的虚□脚本， □□它的信息将取决于所提供的附加参数。从□里出□：

```
#!/bin/sh

. /etc/rc.subr

name="dummy"
start_cmd="${name}_start"
stop_cmd=":"
kiss_cmd="${name}_kiss"
extra_commands="kiss"

dummy_start()
{
        if [ $# -gt 0 ]; then ①
                echo "Greeting message: $*"
        else
                echo "Nothing started."
        fi
}

dummy_kiss()
{
        echo -n "A ghost gives you a kiss"
        if [ $# -gt 0 ]; then ②
                echo -n " and whispers: $*"
        fi
        case "$*" in
        *[.!?])
                echo
                ;;
        *)
                echo .
                ;;
        esac
}

load_rc_config $name
run_rc_command "$@" ③
```

能注意到脚本里□生了那些□□性改□□？

□ □□入的所有在 `start` 之后的参数可以被当作各自方法的定位参数一□被□□。 我□可以根据我□的任□、技巧和想法来以任何方式使用他□。 在当前的例子中，我□只是以下行中字符串的形式□□参数□ `echo(1)` 程序 - 注意 `$*` 是有双引号的。□里是脚本如何被□用的：

```
# /etc/rc.d/dummy start
Nothing started.

# /etc/rc.d/dummy start Hello world!
Greeting message: Hello world!
```

□ 同□用于我□脚本提供的任何方法，并不□限于□准的方法。 我□已□添加了一个自定□的叫做 `kiss` 的方法，并且它□附加参数□来的□□决不□于 `start`。 例如：

```
# /etc/rc.d/dummy kiss
A ghost gives you a kiss.

# /etc/rc.d/dummy kiss Once I was Etaoin Shrdlu...
A ghost gives you a kiss and whispers: Once I was Etaoin Shrdlu...
```

□ 如果我□只是□□所有附加参数□任意的方法， 我□只需要在脚本的最后一行我□□用 `run_rc_command` 的地方，用 `"$@` 代替 `"$1"` 即可。

一个 `sh(1)` 程序□□□是可以理解 `$*` 和 `$@` 的微妙区□只是指定全部定位参数的不同方法。□于此更深入的探□，可以参考□个很好的 `sh(1)` 脚本□程手册。在□完全理解□些表□式的意□之前□不要使用它□， 因□□用它□将□脚本引入缺陷和不安全的弊端。

□在 `run_rc_command` 可能有个缺陷， 它将影□保持参数之□的原本□界。也就是，□有嵌入空白的参数可能不会被正□□理。□缺陷是由于□ `$*` 的□用。

9. □—□□□

Luke Mewburn 的原始文章 中□述了 rc.d 的基本概要， 并□□□述了其□□方案的原理。□文章提供了深入了解整个 rc.d 框架以及其所在的□代 BSD 操作系□的内容。

在 `rc(8)`，`rc.subr(8)`， □有 `rcorder(8)` 的□机手册中，□ rc.d □件做了非常□□的□□。 在□写脚本□，如果不去学□和参考□些□机手册的□， □是无法完全□□出 rc.d 的能量的。

工作中□□□例的主要来源就是□行的系□中的 /etc/rc.d 目□。 它的内容可□性非常好，因□大部分的枯燥的内容都深藏在 `rc.subr(8)` 中了。切□ /etc/rc.d 的脚本也不是神仙写出来的， 所以它□可能也存在着代□缺陷以及低□的□□方案。 但□在□可以来改□它□了！