第 9 章 Nagios 专业话题

9.1. 趣事与玩笑

跟标准的监控程序不一样，Nagios 可以做些很有趣的事情。与其花费时间玩 Quake，为何不花点时间看看这个 http://www.nagios.org/docs/hacks...

9.2. 分布式监控

9.2.1. 介绍

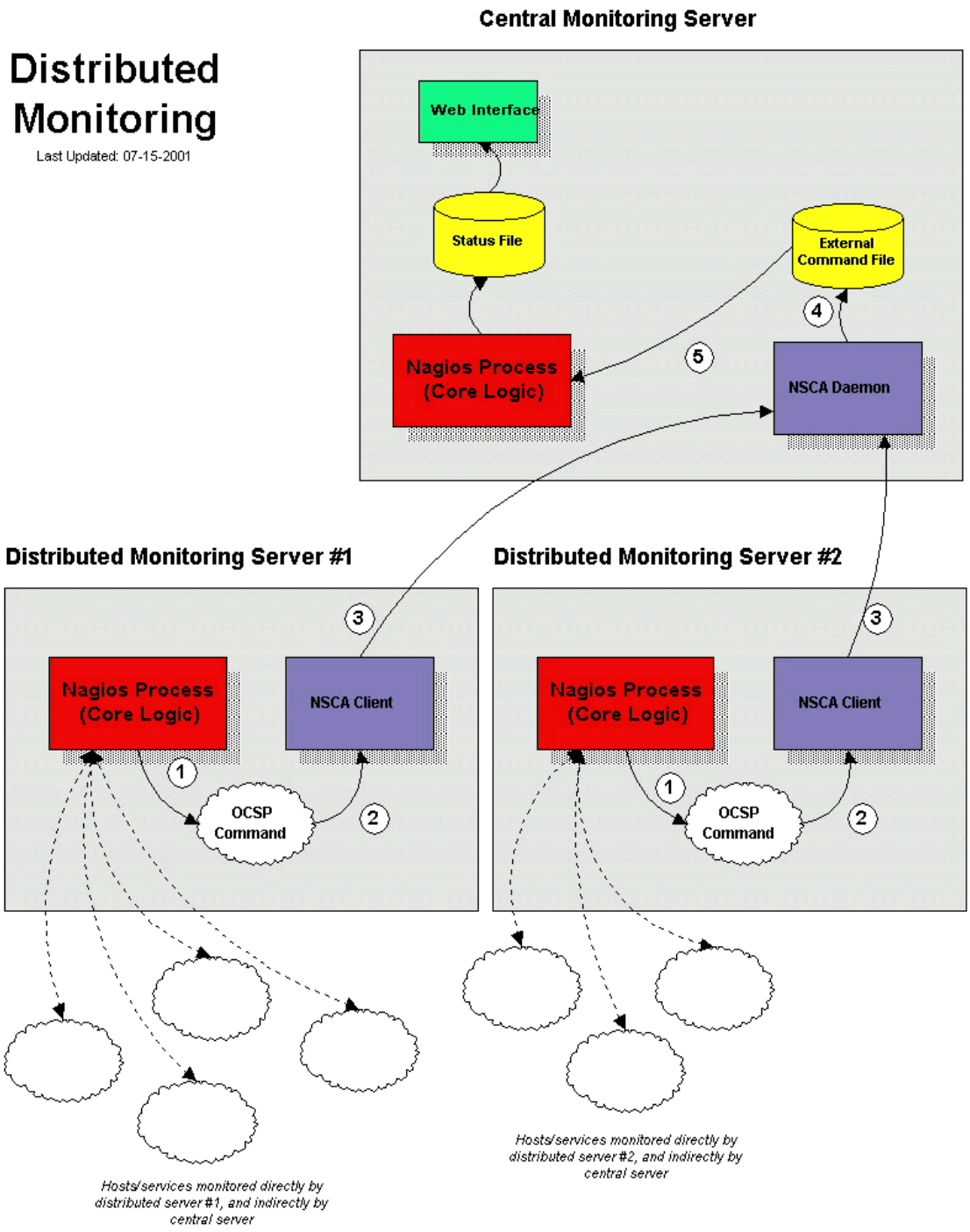
Nagios 可以配置为分布式监控网络服务与资源。下面将尽可能详细阐述如何实现...

9.2.2. 目标

安装一个分布式监控环境的主旨是要降低整体消耗(CPU 利用率等)，通过一个或多个"分布"服务器检测并将结果送给"中心"监控服务器来实现这一目的。不少小型或中型的系统将不会用到这种环境，然而，如果要 Nagios 监控上百台的主机乃至上千台**主机**(和几倍于它的服务个数)时，这就变得非常重要了。

9.2.3. 参照示意图

下图将帮助理解 Nagios 的分布式监控环境如何工作。将利用下图来解释一些概念...



9.2.4. 中心服务与分布服务的对比

当用 Nagios 建立一个分布监控环境时，在中心与分布服务器的配置方面有很大不同。下面将会给出两者的配置并指出在整体上这种配置的影响。先来解释一下这两种位置上不同的服务目标...

"分布"服务器的 功能是真正地完成你所划分出一"组"主机的检测工作。这里的"组"定义是松散的一完成基于你的网络情况而自然形成的。在一个物理位置里可能会有若干个" 组"，这取决于你的网络层次划分，要么因为 WAN 而划分开，要

么因个自独立的防火墙而划分开。很重要的一点是，在每个"组"里都只有一个运行 Nagios 的服务器并完成对该"组"的监控检测工作。分布服务器通常上面只安装有 Nagios，它不需要安装 Web 接口，如果不想让它来做也可以不送出通知、运行事 件处理脚本或是执行任何其他服务检测。有关分布服务器更详尽的内容将在下面配置中给出...

"中心"服务器的目标是从一个或多个分布式服务器收集服务检测结果。虽然中心服务器偶尔也会做些自主检测，但自主检测更多只是在极端情况下才做的，因而可以说中心服务器当前只做强制检测。既然中心服务器从一台或多台分布服务器收集强制服务检测结果，那它就承担全部监控逻辑的整体输出工作(如送出通知、运行事件处理脚本、判定主机状态、安装并提供 Web 接口等)。

9.2.5. 从分布服务器上收集服务检测信息

在研讨配置细节之前，需要了解如何将分布服务器上的服务检测结果送到中心服务器。前面已经讨论过如何提交由 Nagios 发出的强制检测结果到该 Nagios 主机(在强制检测一文中有说明)，但并没有给出任何关于提交不同主机强制检测结果的信息。

为完成从远程主机提交强制检测结果，特意编写了 NSCA 外部构件。 该外部构件包括两部分，第一部分是客户端程序(send_nsca)，运行于远程主机上并负责将强制检测结果送到指定的服务器上去，另一部分是 NSCA 守 护进程(nsca)，它既可以独立地运行于守护服务也可以注册到 inetd 里作为一个 inetd 客户程序来提供监听联接。从客户端收到服务检测结果信息之 后，守护进程将结果提交给在中心服务器的 Nagios，方式是通过在外部命令文件里插入一条 PROCESS_SVC_CHECK_RESULT 命令，之后跟上检测结果。在 Nagios 下一次处理外部命令时将会找到这条由分布式服务器送来的强制检测信息并处理它。很简单对吧？

9.2.6. 分布式监控服务的配置

那么如何来配置一台分布式的 Nagios 服务器？基本要求是一个纯净安装的 Nagios，没必要安装 Web 接口或通知送出服务，因为它们都由中心服务器来完成。

配置的关键差异：

1. 在分布式服务器的对象配置文件里只定义那些由它直接监控的主机与服务的对象；
2. 将分布式服务器的 enable_notifications 域设置为 0，这将阻止它直接送出任何通知信息；
3. 分布式服务器被配置为强迫型服务(obsess over services)类型；
4. 分布式服务有一个强迫型服务处理命令(OCSP)的定义(下面有说明)。

为使网络内的信息充分汇总处理，需要将每个分布服务器上的**全部服务检测**结果送到中心 Nagios 服务器。可以用事件处理来报告服务状态的**变换情况**，但那只是没剪裁的。为强制让分布服务器提交全部的服务检测结果，必须使能位于主配置文件里的强迫型服务 obsess_over_services 选项并且设定好一个强迫型服务处理命令 ocsp_command 以在每次服务检测完成后执行该命令。将利用强迫型服务处理命令来从分布服务器向中心服务器送达全部服务检测结果，这中间将利用 send_nsca 客户端和 nsca 守护服务(上面已经说明)来进行数据传输过程。

为实现这一目标，需要象下面这样定义一个强迫型服务处理(ocsp)命令：

```
ocsp_command=submit_check_result
```

这个 submit_check_result 命令的定义会象是这样：

```
define command{

        command_name    submit_check_result

        command_line
        /usr/local/nagios/libexec/eventhandlers/submit_check_result
$HOSTNAME$ '$SERVICEDESC$' $SERVICESTATE$ '$SERVICEOUTPUT$'

        }
```

该 submit_check_result 的 SHELL 脚本的内容象是这样(用**中心服务器** IP 地址替换里面的 central_server)：

```
#!/bin/sh

        # Arguments:

        #  $1 = host_name (Short name of host that the service is

        #       associated with)

        #  $2 = svc_description (Description of the service)

        #  $3 = state_string (A string representing the status of

        #       the given service - "OK", "WARNING", "CRITICAL"

        #       or "UNKNOWN")

        #  $4 = plugin_output (A text string that should be used
```

```
#       as the plugin output for the service checks)
#


# Convert the state string to the corresponding return code
return_code=-1


case "$3" in
            OK)
            return_code=0
                ;;
        WARNING)
            return_code=1
                ;;
        CRITICAL)
            return_code=2
                ;;
        UNKNOWN)
            return_code=-1
                ;;
esac
# pipe the service check info into the send_nsca program, which
# in turn transmits the data to the nsca daemon on the central
# monitoring server
```

```
        /bin/printf "%s\t%s\t%s\t%s\n" "$1" "$2" "$return_code" "$4" |
/usr/local/nagios/bin/send_nsca central_server -c
/usr/local/nagios/etc/send_nsca.cfg
```

上面脚本中假定已经有 send_nsca 客户端程序，它放在/usr/local/nagios/bin/目录里，并且把配置文件(send_nsca.cfg)放在/usr/local/nagios/etc/目录里。

就这么多！现在已经算是把一个远程的 Nagios 服务器配置成为一个分布服务器了。看一下在分布服务器上到底发生了什么并且它是如何将服务检测结果送到 Nagios 中心服务器上的(下述步骤与前面的参考图中的数字相对应)：

1. 在分布服务器完成一次服务检测后，它会执行所定义的强迫型服务处理命令，就是那个 ocsp_command 变量指向的命令。在本例中，指向的是 **/usr/local/nagios/libexec/eventhandlers/submit_check_result** 脚本。注意在 **submit_check_result** 命令中将四个信息传给了脚本：服务所绑定的主机名、服务描述、服务检测返回结果以及服务检测的输出正文；
2. 该 **submit_check_result** 脚本将服务检测信息(主机名、服务描述、服务检测结果和输出正文)导入到 **send_nsca** 客户端程序；
3. 由 **send_nsca** 程序传送服务检测信息到位于中心服务器上的 **nsca** 守护进程；
4. 位于中心服务器上的 **nsca** 守护进程接收到服务检测信息后写入到外部命令文件里让 Nagios 服务来后续处理；
5. 位于中心服务器的 Nagios 服务进程读入外部命令文件并处理这些来自于远程分布式服务器上的强制服务检测信息。

9.2.7. 中心服务器配置

已经看过分布式服务器如何配置，下面回到中心服务器来做配置。为保障集中处理运行，中心服务器通常只在一台独立运行 Nagios 机器上做配置工作，设置过程如下：

1. 中心服务器安装并配置 Web 接口(可选的，但推荐这么做)；
2. 将中心服务器上的 enable_notifications 域设置为 1，这将使能通知功能(可选的，但推荐这么做)；
3. 将中心服务器的自主服务检测功能关闭(可选的，但推荐这么做－见下面的提示)
4. 使能中心服务器的外部命令检查项(必须的)；
5. 使能中心服务器的强制服务检测(必须的)；

在配置中心服务器时，有另外三件重要的事情要记得做：

1. 中心服务器里必须要有**全部分布服务器上的全部服务**定义。如果没有定义服务对象，在中心的 Nagios 将会忽略那些没有正确配置的强制检测服务的检测结果；
2. 如果中心服务器只负责处理来自分布服务器的服务检测结果，可以在程序层面关闭自主检测，设置 execute_service_checks 域值为 0 即可。如果需要设置中心服务器做些自主检测来监控一些它自己专属的服务(不归分布服务器管)，在分布服务器里这

些服务对象的定义里的 **enable_active_checks** 选项值设为 0。这将阻止 Nagios 对这些服务做自主检测。

很重要的是要么关闭程序层面的自主检测设置，要么把分布服务器里的服务对象定义里的 enable_active_checks 选项设为 0，这将确保自主服务检测不会在一般状况下被执行。这些服务将会以正常的检测周期间隔(3 分钟或 5 分钟等)来重制订计划表，但不会被真的执行。在 Nagios 运行过程中这种重制订计划的循环会不断地重复，后面我会解释一下为何要这么做。

就这么多，很容易，是吧？

9.2.8. 强制检测中的问题

一般意义上讲，中心服务器将只用强制检测方式来做监控。完全依赖于强制检测来做监控的主要问题是 Nagios 必须依赖于其他东西来提供监控系统数据。如果发送强制检测结果的远程主机宕机或不可达时会怎么样？如果 Nagios 不自主检测主机服务，它怎么会知道有故障发生了？

幸运的是有办法来处理这种类型的故障...

9.2.9. 刷新检测

Nagios 支持对服务的检测结果做刷新检测的特性。更多的有关刷新检测信息可以在这篇文档查 看。该特性可以在那些远程主机可能停止送出强制服务检测结果的地方提供针对中心服务器提供保护。"刷新检测"的目的就是要确保服务检测要么可由分布式服务 器以规格化的方式提供检测数据要么由其中心服务器在必要情况下自主地进行检测。如果分布式服务器所提供检测结果被判定"陈旧"，Nagios 将被配置为自 中心监控服务器强制地对那个服务发出自主检测。

那么如何来做呢？在中心服务器上需要对那些由分布式服务器所负责监控的服务做如下配置改动...

1. 服务对象定义里的 **check_freshness** 选项设为 1，这将打开针对该服务的"刷新检测"特性；
2. 服务对象定义里的 **freshness_threshold** 选项须设定为一个以秒为单位的数值，该值反应出由分布式服务器所提供的检测数据将应该以什么样频度来提供出来；
3. 在对象定义里的 **check_command** 选项应指向一个有效的命令，当中心监控服务器需要执行自主检测时可以用此命令来执行检测。

Nagios 定期地对那些打开了"刷新检测"服务的检测结果进行刷新情况检查。服务对象定义里的 freshness_threshold 选项指定了服务检测结果应该在何时间内刷新。例如，如果某个服务里这个选项值是 300，Nagios 将会对当前时间 300 秒(即 5 分钟)之前的检测结果判定为"陈旧"。如果没有指定服务对象里的

freshness_threshold 值，Nagios 将自动地计算出一个刷新间隔门限，要么按照 normal_check_interval 要么按 retry_check_interval 来计算，这取决于服务当时所在什么样的状态类型。一旦服务检测结果被判定为是"陈旧"，Nagios 将使用服务定义里 check_command 指定的命令来执行一次服务检测，这当然是自主服务检测。

一定要记得在中心服务器的服务对象定义里指定 check_command 选项以便进行从中心服务器发出自主服务检测命令。一般情况下，该检测命令不会被执行(因为自主检测在程序层面被关闭了或者是在服务对象定义里关闭了自主检测)。当对结果的刷新检测功能打开时，Nagios 将会运行自主检测命令**既便是自主检测被程序层面被关闭或是服务对象里被关闭也会做(自主检测)**。

如果无法在中心监控主机上定义出针对服务的自主检测命令(也或许是超出了责任范围)，可以在服务定义的 check_command 选项里设一个只是返回紧急状态的脚本来简单地填补该命令。这有个例子...假定定义了一个名叫'service-is-stale'的命令并在服务对象定义里加在了 check_command 选项值里，这个定义可能看起来象这样子...

```
define command{

        command_name    service-is-stale

        command_line    /usr/local/nagios/libexec/staleservice.sh

        }
```

这个 staleservice.sh 脚本在放在/usr/local/nagios/libexec 目录下，内容可能是这样的：

```
#!/bin/sh

/bin/echo "CRITICAL: Service results are stale!"

exit 2
```

当Nagios 发现并判定服务检测结果是陈旧的并运行 service-is-stale 命令来做自主检测时，/usr/local/nagios/libexec/staleservice.sh 脚本将执行并返回一个紧急状态，通常情况下将会引发送出一个故障通知动作，那么这样就可以知道有故障产生了。

9.2.10. 执行主机检测

此时已经知道了如何从分布服务器获取服务的强制检测结果，这意味着中心服务器不会用自主服务检测机制来取得结果，但怎么来做主机检测呢？仍旧需要做主机检测，但如何来做呢？

即然主机检测一般可以折中地只做一小部分自主监控(除非有必要一般是不做的)，我推荐让中心服务器来做点主机自主检测。也就是说，在中心服务器上定义做主机自主检测，这个与分布式服务器上的相同(一般情况下是完全一样的，就象没有做分布式监控一样)。

也可以使用强制主机检测(阅读这篇文档)， 在分布式监控环境下也可以利用这一功能特性，但这可能会导致一些内在问题。最大的问题是 Nagios 将无法正确地处理强制主机检测结果里的故障状态(宕机 与不可达)。这就是说如果监控服务器上的父子节点关系不同(通常情况下因为所处位置不同会有所不同)时，中心服务将会得到一张错误主机当前状态的视图。

如果真的需要从分布式服务器发送强制主机检测结果给中心服务器，确定要做如下配置：

1. 中心服务器里打开强制主机检测开关(必须的)；
2. 分布服务器打开强迫型主机处理开关；
3. 分布服务器里定义好了强迫型主机处理命令(ochp)。

这个强迫型主机处理命令(ochp)被用于处理主机检测结果，与强迫型服务处理命令(ocsp)的执行方式一样，这个方式在前面已经说明过了。为了保存强制主机检测结果即时更新，还要打开主机的刷新检测的功能开关(这个与前面所讲的服务刷新检测相似)。

9.3. 冗余式与失效式网络监控

9.3.1. 介绍

本文档介绍几个在几种类型的网络层上实现冗余主机监控的场景。利用冗余主机，可在运行 Nagios 的核心主机宕机或是一部分网络变为不可达等等情况发生时维持对网络的监控。

注意

如果只是学习如何使用 Nagios，建议你先不要去尝试冗余方式运行，直到你真正熟悉所列出的冗余方式运行所应有的先决条件。冗余式运行有些难以理解而且比较难以正确地配置实现。

9.3.2. 索引

- 先决条件
- 样例脚本

- 场景 1－冗余监控
- 场景 2－失效监控

先决条件

在用 Nagios 实现冗余监控之前，需要熟悉以下内容...

1. 主机与服务事件处理的实现；
2. 用脚本来解决 Nagios 的外部命令问题；
3. 在远程主机上执行插件，要么用 NRPE 外部构件要么是其他的方式；
4. 用 **check_nagios** 插件来检测 Nagios 进程的状态。

样例脚本

本文档里所用脚本可以在 Nagios 发行包的 eventhandlers/子目录里找到。可以修改这些脚本以适合你的监控系统...

场景 1－冗余监控

介绍

在网络中这是一个很容易实现的冗余监控主机的方法，也很初级，它只是对有限数量的故障起到防止作用。为更巧妙地实现冗余监控需要更复杂些的安装配置，更有效的冗余监控是利用不同的网段等方式来实现冗余。

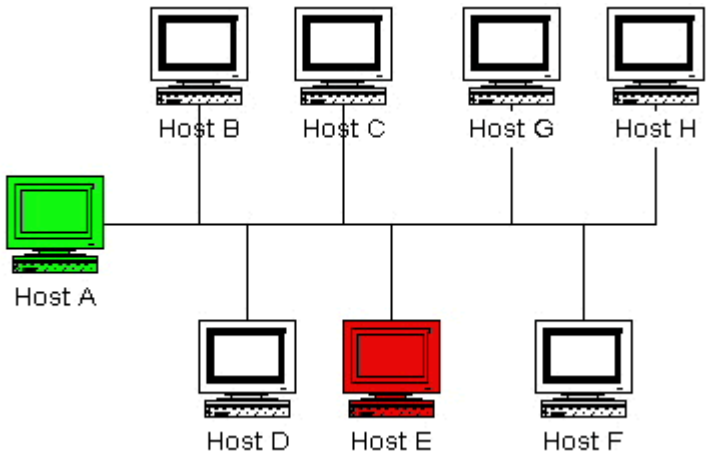
目标

这种冗余形式只是简单实现冗余。把监控"主"机和"从"机针对网络里相同的主机与服务进行监控。一般情况下只是由"主"机把有关故障信息以通知形式送给联系人。希望由"从"机运行 Nagios 并在如下情况发生时完成把故障通知给联系人的工作：

1. 运行 Nagios 的监控"主"机宕机
2. 因某种原因"主"机上的 Nagios 进程停止运行

网络层示意图

下图给出非常简单地网络安装实现的方式。在该场景，假定主机 A 和主机 E 都运行 Nagios 并且都监控全部主机。机器 A 认定是监控"主"机而机器 E 是"从"机。



初始化程序设置

"从"机(机器 E)初始化它的 enable_notifications 域是非使能的，这样阻止了它因任何主机与服务问题而送出通知。还要保证"从"机把 check_external_commands 域使能，这就很容易地实现了...

初始化配置

下面需要考虑在"主"机和"从"机上的对象配置文件差异...

假定已经在"主"机(机器 A)上设置好针对图中全部的主机和服务进行监控。"从"机(机器 E)也需要做相同的设置，还需要做些额外的配置修改...

1. 针对机器 A 的对象定义(在机器 E 的配置文件里)需要给定一个主机的事件处理定义，把该事件处理称为 handle-master-host-event;
2. 机器 E 的配置文件需要定义一个服务来监控机器 A 上 Nagios 服务进程的状态。假定是由机器 A 上的 **check_nagios** 插件来完成这个服务检测工作。可以查阅这个 FAQ 里的描述方法来实现它(需要更新！);
3. 在机器 A 上的配置里针对 Nagios 进程的服务定义时需要有一个事件处理定义，把该事件处理称为 handle-master-proc-event;

注意一点，机器 A("主"机)对机器 E("从"机)一无所知，在该场景里只是没这个必要。当然也可以让机器 A 来监控机器 E，但这这与冗余监控一点关系都没有...

事件处理命令的定义

需要停顿一下来说明在"从"机上的事件处理命令应该是什么样子的，这有个例子...

```
define command{

        command_name    handle-master-host-event
```

```
        command_line
        /usr/local/nagios/libexec/eventhandlers/handle-master-host-ev
ent $HOSTSTATE$ $HOSTSTATETYPE$

        }

define command{

        command_name    handle-master-proc-event

        command_line
        /usr/local/nagios/libexec/eventhandlers/handle-master-proc-ev
ent $SERVICESTATE$ $SERVICESTATETYPE$

        }
```

假定已经把事件处理脚本放在了/usr/local/nagios/libexec/eventhandlers 目录里，也可以把脚本放在其他你想放的位置上，但需要对这些例子做些修改才能用。

事件处理的脚本

OK，看一下事件处理脚本的内容...

主机事件处理脚本(handle-master-host-event)：

```
#!/bin/sh

# Only take action on hard host states...

case "$2" in

HARD)

        case "$1" in

        DOWN)

                # The master host has gone down!

                # We should now become the master host and take

                # over the responsibilities of monitoring the

                # network, so enable notifications...
```

```
        /usr/local/nagios/libexec/eventhandlers/enable_notifications
                ;;
        UP)
                # The master host has recovered!
                # We should go back to being the slave host and
                # let the master host do the monitoring, so
                # disable notifications...
        /usr/local/nagios/libexec/eventhandlers/disable_notifications
                ;;
        esac
        ;;
esac
exit 0
```

服务事件处理脚本(handle-master-proc-event)：

```
#!/bin/sh
# Only take action on hard service states...
case "$2" in
HARD)
        case "$1" in
        CRITICAL)
                # The master Nagios process is not running!
                # We should now become the master host and
```

```
                # take over the responsibility of monitoring
                # the network, so enable notifications...

        /usr/local/nagios/libexec/eventhandlers/enable_notifications
                ;;
        WARNING)
        UNKNOWN)
                # The master Nagios process may or may not
                # be running.. We won't do anything here, but
                # to be on the safe side you may decide you
                # want the slave host to become the master in
                # these situations...
                ;;
        OK)
                # The master Nagios process running again!
                # We should go back to being the slave host,
                # so disable notifications...

        /usr/local/nagios/libexec/eventhandlers/disable_notifications
                ;;
        esac
        ;;
esac
exit 0
```

这样会有什么效果？

"从"机(机器 E)初始不发通知，因而在"主"机(机器 A)上的 Nagios 进程运行时不会送出任何有关主机与服务的通知。

在"从"机上的 Nagios 进程将在下列情况时会变为"主"监控状态...

1. "主"机(机器 A)宕机并且 **handle-master-host-event** 主机事件处理命令脚本被执行；
2. "主"机(机器 A)上的 Nagios 进程停止运行并且 **handle-master-proc-event** 服务事件处理命令脚本被执行；

当"从"机(机器 E)上的 Nagios 进程打开了通知开关时，它将送出任何一个有关主机与服务的故障与恢复的通知，在这一点上，机器 E 扮演着把主机与服务故障通知给联系人的后备冗余的角色！

机器 E 上的 Nagios 进程会重新回到"从"机状态，当如下情况发生时...

1. 机器 A 恢复并且 **handle-master-host-event** 主机事件处理脚本命令被执行；
2. 机器 A 上的 Nagios 进程服务恢复并且 **handle-master-proc-event** 服务事件处理脚本命令被执行；

当机器 E 上的 Nagios 进程的通知功能关闭时，它将不会送出任何有关主机与服务故障与恢复的通知。在这一点上，机器 E 扮演着机器 A 的把主机与服务故障通知到联系人的后备冗余的角色。当系统起动后，所有这些已经各就各位了！

切换时间差

Nagios 的冗余方式并不完善，一个显而易见的问题是在"主"机宕机和"从"机接管之间存在切换时间差。它受如下因素影响：

1. 从"主"机宕机到"从"机首次发现一个故障之间的时间；
2. "主"机真正地验证了一个故障存在(在从机上通过多次服务与主机检测的重试验证)所需时间；
3. 事件处理所需要的执行时间和下一次 Nagios 检查外部命令文件的时间；

可以减小切换时间差，通过修改这些内容...

1. 确保在机器 E 上以较高频度执行对 Nagios 服务的一个或多个检测。可以修改服务定义里的 **check_interval** 和 **retry_interval** 选项来实现；
2. 确保在机器 E 上快速地完成针对机器 A 的重试检测。可以修改主机定义里的 **max_check_attempts** 选项来实现；
3. 在机器 E 上增加外部命令检测的处理频度。可以修改在主配置文件里的 command_check_interval 选项来实现；

当机器 A 上的 Nagios 服务恢复时，同样会有一个切换时间差，完成机器 E 回到"从"机的运行状态，它受如下因素影响：

1. 从机器 A 恢复到机器 E 上的 Nagios 服务检测到服务已经恢复所需时间；
2. 在机器 A 上执行事件处理后的那一时刻起到机器 E 上的 Nagios 服务最近一次处理外部命令文件之间的时间；

监控的冗余监控运行状态的真正切换时间差取决于定义有多少服务、服务以什么间隔被检测和一系列偶然因素。无论如何，这总比没有要好。

特例

有 件事需要注意...如果机器 A 宕机了，机器 E 将使能通知功能并且接管了故障通知。当机器 A 恢复了，机器 E 将关闭通知功能。如果当机器 A 恢复了但是机器 A 上 的 Nagios 服务并没有正确地启动起来话，将会有段时间，在这段时间内不会有任何主机故障的通知被送给联系人！幸运的是，Nagios 的服务检测逻辑会处理这种情况，在机器 E 上的 Nagios 在最近一次的处理机器 A 的状态时，它将发现机器 A 的 Nagios 服务没有运行。机器 E 将会再次打开通知功能并接管 对联系人的故障通知的工作。

统计哪一个主机完成针对网络监控的真正时间是很困难的，很明显，可以通过增高针对机器 A 上的 Nagios 服务的检测频度(在机器 E 上配置)来最小化这一时间间隔，另外就完全是偶然因素的，但整体上的"阻塞"时间并不太多。

场景 2－失效性监控

介绍

失效性监控与冗余监控相似，但有很少一点不同(冗余监控在场景 1 讨论)。

目标

失效监控的最基本目标是"从"机上的 Nagios 机器会在"主"机上的 Nagios 运行时一直保持空转。如果"主"机上的 Nagios 进程服务停止(或者机器宕机)时，"从"机上的 Nagios 将会开始对全部网络监控的工作。

虽然在 While the method described in 场景 1 里所描述的方法将会在"主"机宕机时不断地收到通知，但会有些毛病。最大的问题是"从"机也会**在"主"机在相同的时间里**对网络中的全部主机和服务执行检测！如果在被监控主机上定义有很多服务时这会给主机产生额外的流量及负荷。这里有绕开这个问题的办法...

初始化程序设置

在"从"机上关闭自主检测和通知功能，通用修改 execute_service_checks 和 enable_notifications 选项来实现。这将使得在"主"机运行主 Nagios 服务正常的情况下阻止"从"机对主机与服务的检测和送出通知。同时要确保在"从"机上打开了 check_external_commands 选项。

主服务进程检测

在"从"机上设置一个定时作业(cron job)来周期性地(比如每分钟)运行一个脚本，该脚本用于对"主"机的 Nagios 服务进行检测(在"从"机上使用 check_nrpe 插件，同时，在"主"机上运行 nrpe 守护服务和 check_nagios 插件)，该脚本将检查由 check_nrpe 插件所返回的结果码。如果返回结果码是个非正常状态，脚本将写入一个切换冗余命令到外部命令文件中以使能通知功能和自主检测。如果插件返回一个正常状态，脚本将往外部命令文件里写一个命令以关闭通知功能和自主检测功能。

通过这么做，将会使得在某个时间内只会有一个 Nagios 监控进程在真正运行，这比用两次监控网络的方式效率更高。

还要注意的是，**没有必要**象是在场景 1 里那样定义主机与服务的事件处理，因为处理方式完全不同。

其他问题

此时已经完成了一个基本的失效监控系统的安装设置，然而，要使它工作更顺畅还有几件事要考虑。

这么设置后的最大问题是，"从"机在其开始监控工作时它并*没有当前任何主机与服务*的状态信息。解决这一问题的一个方法是在"主"机上使能强迫型服务处理命令并且把全部的服务检测结果用 nsca 外部构件送 到"从"机上。"从"机就会在承担监控工作时已存有所有服务的最新状态信息。因为"从"机上没有打开自主服务检测，它将不会自主地运行任何服务检测。然 而，它将在必要时执行主机检测。也就是说，"主"和"从"都会在必要时执行主机检测，这不是大问题，因为监控的主要工作大部分是服务的检测。

9.4. 大型安装模式的变化

9.4.1. 介绍

用户在使用 Nagios 大型安装模式将会有许多好处，使用 use_large_installation_tweaks 配置选项。使能这个选项将使 Nagios 守护程序将进行某些短路以使系统负载更低且性能最好。

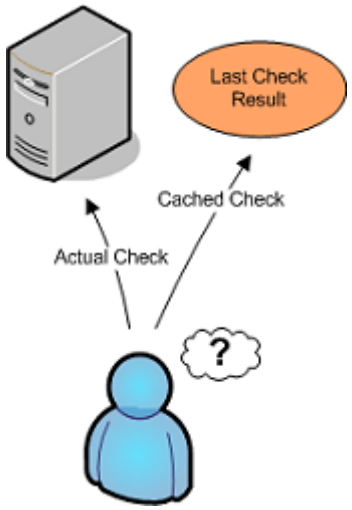
9.4.2. 影响

当你在主配置文件中使能了 use_large_installation_tweaks 配置选项，将会使 Nagios 守护进行做如下变化：

1. 在环境中不能使用汇总类的宏－汇总宏在环境中将不能使用。这些宏的计算在大型安装时会非常地集中消耗时间，因此它们在此时是不能使用的。但如果你在脚本中传递这些参数时，这些汇总性的宏仍旧可用于规格化的宏而加入脚本。
2. 内 存清理有所不同－通常 Nagios 在子进程退出时会释放子进程分配的内存，这是个好习惯，然后在许多安装模式下并不需要，因为许多操作系统将会很小心地处 理进程退出时的内存。操作系统倾向于自主地释放内存而不是由 Nagios 来做，这样更快，因而 Nagios 不再试图释放子进程的内存空间，如果你使能了这 个配置选项的话。
3. 派生 fork()检查更少－通过 Nagios 会在主机和服务检测时做两次派生。这样做是因为(1)确保受阻的插件有一 个较高的进程等级捕获错误信号或进入异常；(2)让操作系统来对那些退出子进程的下级进程做清除处理。额外的派生并不是真有必要，所以在你使能这个配置选 项后它会跳过额外派生动作。Nagios 将自行清理子进程的退出(而不是等到操作系统来做它)这使得 Nagios 在这种安装模式下显著地降低负载。

9.5. 缓存检测

9.5.1. 介绍



应用了缓存检测机制可以显著地改善 Nagios 监控逻辑的性能。缓存检测的作用是，当 Nagios 发现可以利用最近一次检查结果来替代这次检测时，Nagios 会放弃执行一次主机与服务的检测。

9.5.2. 只为按需检测使用

应用缓存检测机制对于通常的规格化编制的主机与服务检测的性能不会有明显改善。缓存检测只是对于主机与服务的按需检测的性能有显著改善。预定的计划性检测可以确保主机与服务的状态更新规范化，它使得在不久的将来，它的检查结果最有可能被缓存检测所利用。

作为参考，要做主机的按需检测...

1. 当绑定于主机上的服务状态发生了变更；
2. 部分检查内容由于需要做主机可达性逻辑判断；
3. 部分是由于需要做主机依赖性检测的前处理。

要做服务的按需检测...

1. 是由于需要做服务依赖性检测的前处理。

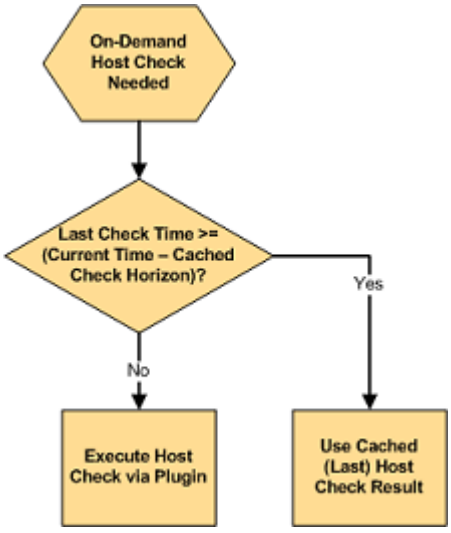
注意

除非你打开了服务依赖性检查，Nagios 将不会使用缓存检测机制来改善服务检测的性能。不必担是，这只是通常情况下是这样。缓存主机检测并不是个极大提升性能做法，每个人都应看到它只是有益于提高性能。

9.5.3. 缓存检测是如何工作的？

当 Nagios 需要做一个主机与服务的按需检测时，它将做一个判定，是否要利用缓存检测结果还是要真的去用插件来做一次检查。这取决于这次主机与服务的最近一次检测结果是否发生于最近的 X 分钟之内，这里 X 是缓存主机与服务结果的时间长度。

如果最近一次检测的时间刚好在指定缓存检测结果的时间内，Nagios 将会利用最近一次针对该主机与服务检测结果而**不会**真的去做一次检测。如果该主机与服务的检测没有做过，或是最近一次检测结果的时间超出缓存检测的时间深度，Nagios 将会用插件对该主机与服务来做一次新的真正的检查。



9.5.4. 这将到底意味着什么？

Nagios 做按需检测是由于它认为有必要及时地知道该主机与服务**在那一时间**里的状态。利用缓存检测结果将使得 Nagios 可以认为最近一次检测结果是"足够好用"的当前主机与服务的状态，并且认定真的没有必要再去做一次该主机与服务的重检测。

缓存检测的时间深度告诉 Nagios 在多长的时间内检测的结果是值得信赖地反应出了当前的主机或服务的状态。比如，时间深度设置是 30 秒，那么在最近的 30 秒之内的主机与服务的检测结果就可以被认为是当前的主机与服务的状态结果。

Nagios 的可用缓存结果数量与需要执行按需检测次数之比被认为是缓存检测的“击中率”。增加缓存检测的时间深度直到该值等于规格化检测的时间间隔，在理论上可以实 现缓存检测的击中率到 100%。在这种情况下，全部的按需检测都可以从缓存检测的结果中提取，多高的性能改善啊!但是真的么？可能并非如此!

缓 存检测结果信息的可信度随时间而降低。高的缓冲击中率需要加长认定为"合法"结果的缓存时间。但各种网络场景变换很快，而且没有任何可以担保在 30 秒之前 处于正常状态的服务当前也是处于正常的。因而不得不取个折中一信任度与速度之间取折中。如果要提高缓存结果的时间深度，就不得不要冒着缓存结果应用于监控 逻辑之中信任度降低的风险。

Nagios 将最终判定出全部主机与服务的正确的状态，因此即便在缓存中的检测结果相对于其真实情况有可能是不 可信的，Nagios 也只是会在一个短时间内在不正确信息之下工作。在这么短时间内的不可信状态信息对于管理员是个讨厌的事情，因为管理员可能会收到故障 通知但它不久就不再有了。

对于 Nagios 用户而言，没有一个标准来检验缓存检测的时间深度或缓存击中率是可接受的。有些需要一个短暂的检 测缓存时间深度设置和一个相对低的缓存击中率，而另一些则想要更长些的缓存时间和较高缓存击中率(当然会相对低的状态可信度)，更有甚者希望完全不用缓存 检测而只要 100%可信度。测试不同的缓存检测时间窗口大小以及对状态信息可信度的影响将只是少数人想做的，他们只想得到在其自身环境下的“正确”取值。 更多的信息见下面讨论。

9.5.5. 配置变量参数

如下的变量将决定用于缓存主机与服务检测结果的时间窗口值，在哪个范围内的检测结果可用于主机与服务的检测结果：

1. cached_host_check_horizon 变量控制缓存主机检测结果；
2. cached_service_check_horizon 变量控制缓存服务检测结果；

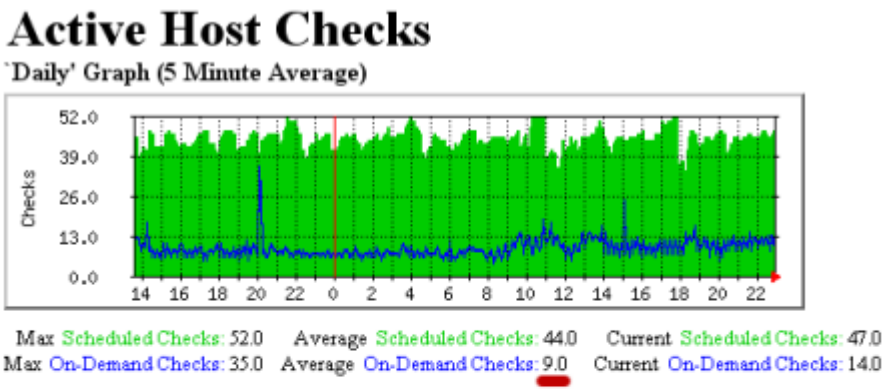
9.5.6. 优化缓存效率

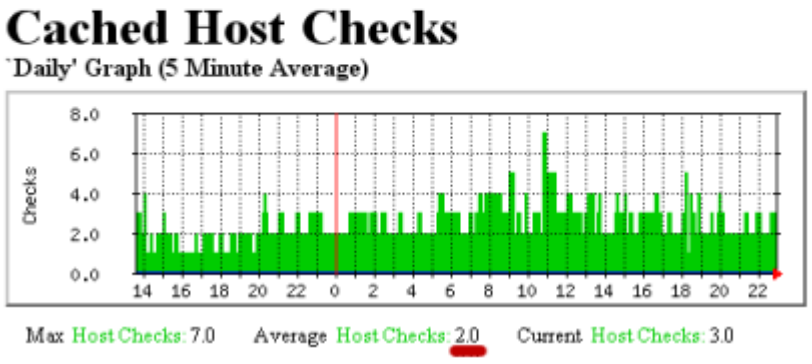
为了应用缓存检测机制达到最高效率，应该做如下工作：

1. 编制规格化的主机检测计划；
2. 使用 MRTG 来绘制统计状态图，做出(1)按需检测图(2)缓存检测图；
3. 调整缓存检测的时间深度以适合当前情况。

在编制主机规格化检测计划时，可以把主机对象定义里的 check_interval 域指定一个大于 0 的值，如果这样做，还应保证将 max_check_attempts 域设置得大于 1，否则会引起一个性能突降，这个性能突降在这篇文档里有说明。

给缓存检测的时间深度取值的一个较好方式是把有多少 Nagios 的按需检测被执行和有多少是取自于缓存检测结果这两个值做比较。nagiostats 工具将提供缓存检测的相关信息，这些信息可以用 MRTG 绘制图表。样例的 MRTG 图表见下面，图中给出了缓存中取结果次数与实际执行检测的次数。





上述监控安装运行而产生图示的事先设置有：

1. 共计有 44 台主机，它们全部用计划检测间隔来检测；
2. 平均(规格化时间表内的)主机检测间隔是 5 分钟；
3. cached_host_check_horizon 值是 15 秒

第一张 MRTG 图表显示了有多少规格化计划主机检测与实际做了多少缓存主机检测的比较。在这例子中，每 5 分钟平均会有 53 次主机检测，其中有 9 次是按需主机检测(占到检测总数的 17%)；

第二张 MRTG 图表显示了沿时间轴上会有多少缓存主机检测结果产生。在这例子中，每 5 分钟平均会有 2 次缓存主机检测；

记 住，缓存检测只是对按需检测起作用。基于图中的每 5 分钟的平均值，可见 Nagios 是每 9 次应做的按需检测中有 2 次是使用缓存检测结果。这看起来不多，但 图中只是给出了一个小型的监控环境的结果，考虑到 2 比 9 就是 22%的性能提高的话，就会明白将会在一个大型监控环境下将会显著地改善性能如果把主机检测的 时间深度加大的话会提高缓存结果的击中率，但也会同时降低了缓存主机状态信息的可信度。

一旦有了几小时乃至几天的 MRTG 图表，就可以看出 主机与服务的检测中有多少是插件执行而有多少是利用的缓存结果。利用这些图表信息来调整缓存检测的时间深度以适合当前环境，不断地利用 MRTG 图表来监视 缓存检测时间深度变量对缓存检测统计在时间维度上的影响情况，并在需要的时候清掉重新来过。

9.6. 状态追踪

9.6.1. 介绍

状 态“追踪”是个并不通用的功能特性。使能了它，即便是主机与服务的状态没有变化的情况下也可对状态检测的结果产生日志记录。当对部分主机与服务开启了状态 追踪功能时，Nagios 将会仔细地监控这些主机与服务并记录下任何有关的检测结果的输出内容。可见，这对于日后分析日志文件很有帮助。

9.6.2. 它是如何工作的？

在通常情况下，只有在主机与服务的状态与最近一次的检测结果发生变化时才会对检测结果产生日志记录。只有很少情况例外，这是个规则。

如果对部分的主机与服务使能一种或多种的状态追踪功能，Nagios 将在本次检测结果与前一次检测有差异的情况下产生输出结果的日志记录。下面例子中有八次连续的服务检测，看看有什么差别：

表 9.1.

服务检测号	服务状态	检测的结果输出	通常日志	有跟踪的日志
x	正常	RAID array optimal		
x+1	正常	RAID array optimal		
x+2	告警	RAID array degraded (1 drive bad, 1 hot spare	✓	✓

服务检测号	服务状态	检测的结果输出	通常日志	有跟踪的日志
		rebuilding)		
x+3	紧急	RAID array degraded (2 drives bad, 1 host spare online, 1 hot spare rebuilding)	√	√
x+4	紧急	RAID array degraded (3 drives bad, 2 hot spares online)		√
x+5	紧急	RAID array failed		√
x+6	紧急	RAID array failed		
x+7	紧急	RAID array failed		

在上述的一系列检测中，一般的日志方式只可以看到对此次网络事故的两条记录。第一条是在 x+2 次时服务从正常变换为告警状态，第二条是在 x+3 时，状态由告警转变为紧急。

无论何原因，你可能需要在日志文件里对此次网络事故保存有完整的历史记录。可能是有助于你向主管解释事故情况失控是何等的快，也可能是在酒巴里与一堆人谈笑有个好话题...

那 么，如果对该服务的紧急状态使能了追踪功能，将会在 x+4 和 x+5 时多两个日志事件。这是为什么呢？当打开了状态追踪功能，Nagios 将对每一次检测结果进行检查，看此次结果与最近一次结果是否存在差异。如果输出状态不同而且服务的状态并没有改变，那么这次新检测结果的输出就会被日志文件记录下来。

状 态追踪最直接的例子是对一个 WWW 服务的监控检测。如果 check_http 检测插件第一次检测时，因一个 404 错误返回一个告警状态，之后因为期望匹配 串没有找到返回一个告警，这可能是你需要知道的情况。如果没有使能 WWW 服务的告警状态追踪，只会得到第一次转入告警状态时日志(404 错误而产生的)， 但自此之后是因为在返回页面里没有匹配到指定字符串而产生的告警状态不是 404 错误，这会让你在打开压缩的日志文件包时不知如何下手。

9.6.3. 需要使能状态追踪么？

首先，必须断定你真的需要在打包的日志文件里寻找故障根源。你可能是只需要对几个主机与服务打开这个功能而非全部。你可能是只需要对某个主机或某个服务的某几个状态进行状态追踪而非全部。比如，你需要对一个服务的告警和紧急状态进行状态追踪但不需要对正常和未知进行追踪。

对部分主机或服务的状态追踪是否打开使能也跟对该主机与服务做检测的插件有关。如果对于某种状态，检测插件总是返回相同的输出字符串，那就没理由为该服务打开状态追踪功能(——打开也没用，全是一个内容)。

9.6.4. 如何使能追踪？

可以在主机与服务对象定义的 stalking_options 域设置状态追踪功能的打开或关闭。

9.6.5. 状态追踪与可变服务有何不同？

状态追踪与可变服务相似。但可变服务会引发通知动作和事件处理的执行，而状态追踪纯粹是为了产生详细日志。

9.6.6. 限制与告诫

需要当心在使用状态追踪时会有一些潜在问题。主要会发生在几个 CGI 模块(历史状态、报警汇总等)的报告功能中。因为状态追踪会增加一些额外的日志，这些会增大产生故障日志的次数，而生成报告时是以这些数据为基础的。

作为一条通行规则，建议在没有考虑完善之前**不要**打开主机与服务的状态追踪功能，除非，你确实想要用这个功能。

9.7. 集群主机和集群服务的监控

9.7.1. 介绍

有些人咨询有关集群主机和集群服务的监控，因而写本文档来解释如何来做，希望这些简单明瞭。

首先来解释一下“集群”。用例子来说明较容易理解。有 5 台主机来提供 DNS 解析服务，如果一台宕机，这不算什么，因为仍旧有几台机器在提供 DNS 服务。如果需要监控这个 DNS 服务，那么就有 5 个 DNS 服务器，这种情况下，这个 DNS 服务被认为是一个**集群服务**，它有 5 个独立的 DNS 服务构成。虽然需要各自独立地对服务进行监控，但更关心的是整体的 DNS 服务集群能否正常工作，而不是某个独立的服务工作情况。

如果你有一个主机群来提供高可靠性服务(集群)解决方案，这种情况被认为是一个**集群主机**。如果一台主机宕机，另一个将接管全部失效主机的工作。一个提示，如果想配置一个 Linux 集群系统，可以访问高可靠性 Linux 集群项目。

9.7.2. Plan of Attack

有几步来监控集群服务和集群主机。下面将尽可能简单地说明。监控集群主机和集群服务有两步要做：

1. 监控集群内的元件
2. 监控一个集合体

监控集群中的主机与服务元件比较容易。事实上可能你已经做过到了。对于集群服务，只要确保已经实现了对集群服务中的每个服务都已经处于监控状态，如果是一个 5 台 DNS 服务组成的集群，要保证定义出了 5 台域名解析服务对象(可能会用到 check_dns 插件)。对于集群主机，要确保对集群主机中的每一台都有对象定义(同样必须要给每个主机至少要绑定一个服务)。

重要

可能要关闭集群主机或集群服务里每个元件的报警通知功能。虽然每个元件没有独立送出通知，但仍旧可以在当前状态 CGI 模块里独立地查看每个集群元件状态显示。这样可能有助于查找集群服务的故障根源。

监控集群整体可以使用对集群每个元件检测的缓存检测结果。虽然也可以在集群检测的时候对全部元件进行再检测，但有缓存结果的情况下为什么要再次浪费带宽和资源来再做一遍呢？缓存结果在哪里？在状态文件中保存了集群中的每个元件的缓存检测结果(假定已经对每个元件进行监控)。那个 check_cluster 插件被设计为在状态文件中取出缓存检测结果来完成对集群检测。

重要

虽然没有对集群每个元件使能通知功能，但仍旧可以完成对集群整体的检测。

9.7.3. 使用集群检测 check_cluster 插件

那个 check_cluster 插件被设计为从集群中每个独立的集群元件状态结果中提供状态信息来生成集群整体状态。

check_cluster 插件可以在 Nagios 插件软件包(http://sourceforge.net/projects/nagiosplug/)的发行目录中找到。

9.7.4. 监控服务集群

假定要对一个由 3 台 DNS 服务组成的了冗余域名解析服务群进行监控。首先，在做为一个集群监控之前，已经可以各自独立地完成对三个域名解析服务的监控，假定是 3 台 DNS 主机分别叫 host1、host2 和 host3，它们上面绑定有名为"DNS Service"的服务。

为了完成对集群监控，需要创建一个"集群"服务对象。然而，在此之前，需要先定义一个对集群服务检测命令配置。假定这个叫做 **check_service_cluster** 命令是这样定义的：

```
define command{

                command_name    check_service_cluster

                command_line
        /usr/local/nagios/libexec/check_cluster --service -l $ARG1$ -w
$ARG2$ -c $ARG3$ -d $ARG4$

                }
```

现在就可以创建一个叫"cluster"的集群服务对象了，里面用刚刚定义的 **check_service_cluster** 命令来做检测。下面例子将示意如何来做。例子中给出了如果有 2 个或 2 个以上的服务处于非正常状态时将会产生一个紧急状态，如果只有 1 个服务处于非正常状态时将产生告警状态，如果全部正常将返回一个集群服务处于"正常"的状态。

```
define service{

                ...

                check_command   check_service_cluster!"DNS
Cluster"!1!2!$SERVICESTATEID:host1:DNS
Service$,$SERVICESTATEID:host2:DNS Service$,$SERVICESTATEID:host3:DNS
Service$

                ...

                }
```

很重要的一点是，要注意在集群检测命令里的第 4 个命令参数宏$ARG4$里使用了一个逗号分隔的**按需**服务状态宏列表，这个非常重要！Nagios 将当前集群中的每个服务状态 ID 来填充这些按需宏的状态值(用数值而不是字符串)，有关按需主机与服务宏的信息可以查阅这篇文档。

9.7.5. 主机集群的监控

对集群主机的监控与集群服务相类似，当然最主要的不同的组成集群的是主机而非服务。为完成对集群主机的监控，必须定义一个使用 check_cluster 插件的服务对象。该服务将**不属于**集群中的任何一个主机，因为绑定在某个主机上时，当它宕机时会使得集群主机的通知无法送出。把这个集群服务绑定在 Nagios 运行主机上是个好主意。毕竟如果 Nagios 运行可以对集群检查，如果宕掉，也就无法进行监控了(除非你设置了冗余监控主机)...

不管如何，假定有一个命令 check_host_cluster 的定义是这样的：

```
define command{

        command_name    check_host_cluster

        command_line
    /usr/local/nagios/libexec/check_cluster --host -l $ARG1$ -w
$ARG2$ -c $ARG3$ -d $ARG4$

        }
```

假设是个 3 台主机(分别命名为"host1"、"host2"和"host3")组成的集群主机。如果想让 Nagios 在发现有 1 台没有运行时送出告警报警，在 2 台或 2 台以上时送出紧急报警，那么集群主机的监控会这样来定义：

```
define service{

        ...

        check_command   check_host_cluster!"Super Host
Cluster"!1!2!$HOSTSTATEID:host1$,$HOSTSTATEID:host2$,$HOSTSTATEID:hos
t3$

        ...

        }
```

很重要的一点是，要注意在集群检测命令里的第 4 个参数$ARG4$是传递了一个逗号分隔的**按需**主机状态宏列表。这个很重要！ Nagios 将会用集群里当前各个主机状态的 ID 值(数值而不是字符串)来填充按需状态宏。

就 这么多。Nagios 将定期地检测集群主机的状态并在状态变换时(假定已经使能了该服务的通知功能)送给你通知。注意对于集群里的每个主机成员，可能需要 关闭主机宕机等的通知功能。记住，对于一个集群来说，你真正关心的一是某台主机的状态而是集群整体状态。你或许想在主机对象定义里去掉有关不可达通知的功 能，这取决于你的网络层情况以及如何现构建集群。

9.8. 适应性监控

9.8.1. 介绍

Nagios 允许你在运行时对主机和服务进行特定检查时变更命令。我把这种特性称为"适应性监控"。请注意 Nagios 的适应性监控对于 99%的用户是不需要的，但可以让你做些有趣的事情。

9.8.2. 什么可以改？

在运行时，如下服务检测属性是可以修改的：

1. 检测命令和命令行参数
2. 检测周期
3. 最大检测尝试次数
4. 检测周期
5. 事件处理命令及命令参数

在运行时，如下主机检测属性是可以修改的：

1. 检测命令和命令行参数
2. 检测周期
3. 最大检测尝试次数
4. 检测周期
5. 事件处理命令及命令参数

在运行时，如下的全局属性可以修改：

1. 全局的主机事件处理命令及命令参数
2. 全局的服务事件处理命令及命令参数

9.8.3. 适应性监控的外部命令

为了在运行时改变全局的、主机的或服务的属性，你需要给出恰当的外部命令给 Nagios，在外部命令文件中设置。表格列出的不同属性可以完成对各自不同的属性进行修改。

一个给适应性检测而制作的完整外部命令列表（包括如何使用样例）可以在如下 URL 中找到：http://www.nagios.org/developerinfo/externalcommands/

注意

注意以下内容：

1. 当修改检测命令、检测周期或事件处理句柄时，很重要的一点是在 Nagios 启动之前要注意给这些新值定义好。如果在 Nagios 启动之后，任何试图修改这些设置的尝试都会被忽略。
2. 你可以给指定特定的命令参数给这些命令名－只是使用分隔符(!)分开几个命令参数。更多的有关如何定义命令参数的信息可以在宏及应用的文档中找到。

9.9. 强制式主机状态迁移

9.9.1. 介绍

当 Nagios 用强制检测方式从远程源接收主机检测结果时（如其他的 Nagios 分布式实例或分散式安装），由远程资源上报告的主机的状态可能并不能正确地 显示在 Nagios 的视图上。在处于分布式或分散式安装方式下由多个 Nagios 实例结果中保证正确地显示主机状态是非常重要的。

9.9.2. 不同的全局视图

下图给出分散式安装的简单例子。图中

1. **Nagios-A** 是主监控服务器并可以对全局的路由器和交换机进行监控。
2. **Nagios-B** 和 **Nagios-C** 是后备的监控服务器，可以从 **Nagios-A** 接收强制检测结果。
3. 当 **Router-C** 和 **Router-D** 处于故障并离线状态。

那么 Router-C 和 Router-D 当前应处于什么状态？结果取决于你访问哪个 Nagios 实例。

1. **Nagios-A** 报告 **Router-D** 处于宕机且 **Router-C** 处于不可达
2. **Nagios-B** 报告 **Router-C** 处于宕机且 **Router-D** 处于不可达
3. **Nagios-C** 报告全部的路由器处于宕机

每个 Nagios 实例都有不同的网络状态视图，由于后备的监控服务不可以盲目地从主监控主服务器接收主机状态否则它们会得不到正确的网络状态信息。

由于没有转换主监控服务器(Nagios-A)的强制主机检测结果，Nagios-C 将认为 Router-D 处于不可达，除非它自己得到其真的宕机。相同地，宕机或不可达状

态(从 Nagios-A)看过去的 Router-C 和 Router-D 的视图会使得 Nagios-B 的视图翻转。

注意

有时你不想让 Nagios 因为远程的源给出的状态而使得视图中显示宕机或不可达状态而翻转你处于“正确”状态的视图，如分布式环境下，你想让中心监控服务器得到不同的分布节点下的不同网络部分的视图。

9.9.3. 使能状态迁移

默认情况下，Nagios 将**不会**自动地用强制检测的宕机和不可达状态来迁移状态。如果你需要必须使能它。

自动地将强制检测结果进行状态迁移受 translate_passive_host_checks 变量的控制。使能它将使本地的 Nagios 实例接收来自远程资源的宕机和不可达状态迁移而改变显示状态。