第 5 章 准备配置 Nagios

5.1. 配置概览

5.1.1. 介绍

在你开始监控网络与系统之前要有同个不同配置文件需要创建和编辑。耐心点，配置 Nagios 可能是要花些时间特别是对于那些初次使用者。弄清其机理所有的将它们搞定绝对是值得的。 :-)

注意

样本配置文件在安装时放在了/usr/local/nagios/etc/目录下，如果你是按照前面给出的快速安装指南来做的话。

5.1.2. 主配置文件

主配置文件包括了一系列的设置，它们会影响 Nagios 守护进程。不仅是 Nagios 守护进程要使用主配置文件，CGIs 程序组模块也需要，因此，主配置文件是你开始学习配置其他文件的基础。

有关主配置文件的文档在这里。

5.1.3. 资源配置文件

资源文件可以保存用户自定义的宏。资源文件的一个主要用处是用于保存一些敏感的配置信息如系统口令等不能让 CGIs 程序模块获取到的东西。

你可以在主配置文件中设置 resource_file 指向一个或是多个资源文件。

5.1.4. 对象定义文件

对象定义文件用于定义主机、服务、主机组、服务组、联系人、联系人组、命令等等。这些将定义你需要监控什么并将如何监控它们。

你可以在主配置文件里设置 cfg_file 加上 cfg_dir 来指向一个或是多个对象定义文件。

有关对象定义和与其他间关系的文档是这里。

5.1.5. CGI 配置文件

CGI 配置文件包含了一系列的设置，它们会影响 CGIs 程序模块。还有一些保存在主配置文件之中，因此 CGI 程序会知道你是如何配置的 Nagios 并且在哪里保存了对象定义。

有关 CGI 配置文件的文档在这里。

5.2. 主配置文件选项

注意

当创建或编辑配置文件时，要遵守如下要求：

- 以符号'#'开头的行将视为注释不做处理；
- 变量必须是新起的一行 － 变量之前不能有空格符；
- 变量名是大小写敏感的；

提示

样例配置文件(/usr/local/nagios/etc/nagios.cfg)已经安装到位，如果你是按照快速安装指南来操作的话。

5.2.1. 配置文件的位置

主配置文件一般(实际是固定的)是 nagios.cfg，存放位置在 /usr/local/nagios/etc/目录里(－－如果是 rpm 包来安装，应该是在 /etc/nagios/)。

5.2.2. 配置文件里的变量

下面将对每个主配置文件里的选项进行说明...

表 5.1. 日志文件

格式：	**log_file=<file_name>**
样例：	**log_file=/usr/local/nagios/var/nagios.log**

这个变量用于设定 Nagios 在何处创建其日志文件。它应该是你主配置文件里面的第一个变量，当 Nagios 找到你配置文件并发现配置里有错误时会向该文件中写入错误信息。如果你使能了日志回滚，Nagios 将在每小时、每天、每周或每月对日志进行回滚。

表 5.2. 对象配置文件

格式：	**cfg_file=<file_name>**
样例：	cfg_file=/usr/local/nagios/etc/hosts.cfg cfg_file=/usr/local/nagios/etc/services.cfg cfg_file=/usr/local/nagios/etc/commands.cfg

该变量用于指定一个包含有将用于 Nagios 监控对象的对象配置文件。对象配置文件中包括有主机、主机组、联系人、联系人组、服务、命令等等对象的定义。配置信息可以切分为多个文件并且用 cfg_file=语句来指向每个待处理的配置文件。

表 5.3. 对象配置目录

格式：	**cfg_dir=<directory_name>**
样例：	cfg_dir=/usr/local/nagios/etc/commands cfg_dir=/usr/local/nagios/etc/services cfg_dir=/usr/local/nagios/etc/hosts

该变量用于指定一个目录，目录里包含有将用于 Nagios 监控对象的对象配置文件。所有的在这个目录下的且以.cfg 为扩展名的文件将被作为配置文件来处理。另外，Nagios 将会递归该目录下的子目录并处理其子目录下的全部配置文件。你可以把配置放入不同的目录并且用 cfg_dir=语句来指向每个待处理的目录。

表 5.4. 对象缓冲文件

格式：	**object_cache_file=<file_name>**
样例：	**object_cache_file=/usr/local/nagios/var/objects.cache**

该变量用于指定一个用于缓冲对象定义复本的文件存放位置。对象缓冲将在每次 Nagios 的启动和重启时和使用 CGI 模块时被创建或重建。它试图加快在 CGI 里的配置缓冲并使得你在编辑对象配置文件时可以让正在运行的 Nagios 不影响 CGI 的显示输出。

表 5.5. 预缓冲对象文件

格式：	**precached_object_file=<file_name>**
样例：	**precached_object_file=/usr/local/nagios/var/objects.precache**

该变量用于指定一个用于指定一个用于预处理、预缓冲 This directive is used to specify a file in which a pre-processed, pre-cached copy of 对象定义复本的文件存放位置。在大型或复杂 Nagios 安装模式下这个文件可用于显著地减少 Nagios 的启动时间。如何加快启动的更多信息可以查看这个内容。

表 5.6. 资源文件

格式：	**resource_file=<file_name>**
样例：	**resource_file=/usr/local/nagios/etc/resource.cfg**

该变量用于指定一个可选的包含有$USERn$宏定义的可选资源文件。$USERn$宏在存放用户名、口令及通用的命令定义内容(如目录路径)时非常有用。CGIs 模块将不会试 图读取资源文件，所以你可以限定这权文件权限(600 或 660)来保护敏感信息。你可以在主配置文件里用 resource_file 语句来加入多个资源文 件－Nagios 将会处理它们。如何定义$USERn$宏参见样例 resource.cfg 文件，它放在 Nagios 发行包的 sample-config/子目录下。

表 5.7. 临时文件

格式：	**temp_file=<file_name>**
样例：	**temp_file=/usr/local/nagios/var/nagios.tmp**

该变量用于指定一个临时文件，Nagios 将在更新注释数据、状态数据等时周期性地创建它。该文件不再需要时会删除它。

表 5.8. 临时路径

格式：	**temp_path=<dir_name>**
样例：	**temp_path=/tmp**

这个变量是一个目录，该目录是块飞地，在监控过程中用于创建临时文件。你应在该目录内运行 tmpwatch 或类似的工具程序以删除早于 24 小时的文件(这是个垃圾文件存放地)。

表 5.9. 状态文件

格式：	**status_file=<file_name>**
样例：	**status_file=/usr/local/nagios/var/status.dat**

这个变量指向一个文件，文件被 Nagios 用于保存当前状态、注释和宕机信息。CGI 模块也会用这个文件以通过 Web 接口来显示当前被监控的状态，CGI 模块必须要有这个文件的读取权限以使工作正常。在 Nagios 停机或在重启动时将会删除并重建该文件。

表 5.10. 状态文件更新间隔

格式：	**status_update_interval=<seconds>**
样例：	**status_update_interval=15**

这个变量设置了 Nagios 更新状态文件的速度(秒为单位)，最小更新间隔是 1 秒。

表 5.11. Nagios 用户

格式：	**nagios_user=<username/UID>**
样例：	**nagios_user=nagios**

该变量指定了 Nagios 进程使用哪个用户运行。当程序启动完成并开始监控对象之前，Nagios 将切换自己的权限并使用该用户权限运行。你可以指定用户或是 UID 名。

表 5.12. Nagios 组

格式：	**nagios_group=<groupname/GID>**
样例：	**nagios_group=nagios**

该变量用于指定 Nagios 使用哪个用户组运行。当程序启动完成并开始监控对象之前，Nagios 将切换自己的权限并以该用户组权限运行。你可以拽定用户组或 GID 名。

表 5.13. 通知选项

格式：	**enable_notifications=<0/1>**
样例：	**enable_notifications=1**

该选项决定了 Nagios 在初始化启动或重启动时是否要送出通知。如果这个选项不使能，Nagios 将不会向任何主机或服务送出通知。注意，如果你打开了状态保持选项，Nagios 在其启动和重启时将忽略此设置并用这个选项的最近的一个设置(已经保存在状态保持文件)的值来工作，**除非**你取消了 use_retained_program_state 选项。如果你想在使能状态保存选项(并且是 use_retained_program_state 使能)的情况下更改这个选项，你必须要通过合适的外部命令或是通过 Web 接口来修改它。选项的取值可以是：

1. 0 = 关闭通知
2. 1 = 打开通知(默认)

表 5.14. 服务检测执行选项

格式：	**execute_service_checks=<0/1>**
样例：	**execute_service_checks=1**

这个选项指定了 Nagios 在初始的启动或重启时是否要执行服务检测。如果这个没有使能，Nagios 将不会主动地执行任何服务的检测并且保持一系列的"静默"状态(它仍旧可以接收强制检测除非你已经将 accept_passive_service_checks 选项关闭)。这个选项经常用于备份被监控服务配置，被监控服务的配置备份在文档冗余安装或设置成一个分布式监控环境中有描述。注意：如果你已经使能了状态保持，Nagios 在其启动或重启时将会忽略这个选项设置并使用和旧的设置值(旧值保存于状态保持文件)，**除非**你关闭了 use_retained_program_state 选项。如果你想在状态保持使能(和 use_retained_program_state 选项使能)的情况下修改这个选项，你只得用适当的外部命令或是通过 Web 接口来修改它。选项可用的值有：

1. 0 = 不执行服务检测
2. 1 = 执行服务检测(默认)

表 5.15. 强制服务检测结果接受选项

格式：	**accept_passive_service_checks=<0/1>**
样例：	**accept_passive_service_checks=1**

该选项决定了 Nagios 在其初始化启动或重启后是否要授受强制服务检测，如果它关闭了，Nagios 将不会接受任何强制服务检测结果。注意：如果你已经使能了状态保持，Nagios 在其启动或重启时将会忽略这个选项设置并使用和旧的设

置值(旧值保存于状态保持文件)，**除非**你关闭了 use_retained_program_state 选项。如果你想在状态保持使能(和 use_retained_program_state 选项使能)的情况下修改这个选项，你只得用适当的外部命令或是通过 Web 接口来修改它。选项可用的值有：

1. 0 = 不接受强制服务检测结果
2. 1 = 接受强制服务检测结果(默认)

表 5.16. 主机检测执行选项

格式：	**execute_host_checks=<0/1>**
样例：	**execute_host_checks=1**

该选项将决定 Nagios 在初始地启动或重启时是否执行按需地和有规律规划检测。如果该选项不使能，那么 Nagios 将不会对任何主机进行检测，然而它仍旧可以接收强制主机检测结果除非你已经将 accept_passive_host_checks 选项关闭。该选项通常用于监控服务器的配置备份，详细信息请查看冗余安装的配置，或是用于设置一个分布式监控环境中。注意：如果你已经使能 retain_state_information 状态保持选项，Nagios 将在启动和重启时使用旧的选项值(保存于 state_retention_file 状态保持文件中)而忽略此设置，**除非**你关闭了 use_retained_program_state 选项。如果你想在保持选项使能(且 use_retained_program_state 选项使能)的情况下修改这个选项，你只得用适当的外部命令或是通过 Web 接口来修改它。选项可用的值有：

1. 0 = 不执行主机检测
2. 1 = 执行主机检测(默认)

表 5.17. 强制主机检测接受选项

格式：	**accept_passive_host_checks=<0/1>**
样例：	**accept_passive_host_checks=1**

该选项决定了在 Nagios 初始启动或重启后是否要接受强制主机检测结果。如果这个选项关闭，Nagios 将不再接受任何强制主机检测结果。注意：如果你使能 retain_state_information 状态保持选项，Nagios 将在启动或重启动时使用旧的选项设置(保存于 state_retention_file 状态保持文件中)而忽略这个设置。**除非**你已经关闭 use_retained_program_state 选项。如果你想在保持选项使能(且 use_retained_program_state 选项使能)的情况下修改这个选项，你只得用适当的外部命令或是通过 Web 接口来修改它。选项可用的值有：

1. 0 = 不接受强制主机检测结果
2. 1 = 接受强制主机检测结果(默认)

表 5.18. 事件处理选项

格式：	**enable_event_handlers=<0/1>**
样例：	**enable_event_handlers=1**

该选项决定了在 Nagios 初始启动或重启后是否要运行事件处理，如果该选项关闭，Nagios 将不做任何主机或服务的事件处理。注意：如果你使能 retain_state_information 状态保持选项(保存于 state_retention_file 状态保持文件中)而忽略这个设置，**除非**你已经关闭 use_retained_program_state 选项。如果你想在保持选项使能(且 use_retained_program_state 选项使能)的情况下修改这个选项，你只得用适当的外部命令或是通过 Web 接口来修改它。选项可用的值有：

1. 0 = 禁止事件处理
2. 1 = 打开事件处理(默认)

表 5.19. 日志回滚方法

格式：	**log_rotation_method=<n/h/d/w/m>**
样例：	**log_rotation_method=d**

该选项决定了你想让 Nagios 以何种方法回滚你的日志文件。可用的值有：

1. n = None (不做日志回滚 — 这个是默认值)
2. h = Hourly (每小时做一次日志回滚)
3. d = Daily (每天午夜做日志回滚)
4. w = Weekly (每周六午夜做日志回滚)
5. m = Monthly (每月最后一天的午夜做日志回滚)

表 5.20. 日志打包路径

格式：	**log_archive_path=<path>**
样例：	**log_archive_path=/usr/local/nagios/var/archives/**

该选项将指定一个用于存放回滚日志文件的保存路径。如果没有使用日志回滚功能时会忽略此设置。

表 5.21. 外部命令检查选项

格式：	**check_external_commands=<0/1>**
样例：	**check_external_commands=1**

该选项决定了 Nagios 是否要检查存于命令文件里的将要执行的命令。这个选项在你计划通过 Web 接口来运行 CGI 命令时必须要打开它。更多的关于外部命令的信息可以查阅这份文档。

1. 0 = 不做外部命令检测
2. 1 = 检测外部命令(默认值)

表 5.22. 外部命令检测间隔

格式：	**command_check_interval=<xxx>[s]**
样例：	**command_check_interval=1**

如果你指定了一个数字加一个"s"(如 30s)，那么外部检测命令的间隔是这个数值以**秒**为单位的时间间隔。如果没有用"s"，那么外部检测命令的间隔是以这个数值的“时间单位”的时间间隔，除非你把 interval_length 的值(下面有说明)从默认 60 给更改了，这个值的意思是 60s，即一分钟。

注意：将这个值设置为-1 可令 Nagios 尽可能频繁地对外命令进行检测。在进行其他任务之前，Nagios 每次都将会读入并处理保存于命令文件之中的全部命令以进行命令检查。更多的关于外部命令的信息可以查阅这份文档。

表 5.23. 外部命令文件

格式：	**command_file=<file_name>**
样例：	**command_file=/usr/local/nagios/var/rw/nagios.cmd**

这是一个 Nagios 用于外部命令检测处理的文件，命令 CGI 程序模块将命令写入该文件，外部命令文件实现成一个命名管道(先入先出)，在 Nagios 启动时创建它，并在关闭时删除它。如果在 Nagios 启动时该文件已经存在，那么 Nagios 会给出一个错误信息后中止。更多的关于外部命令的信息可以查阅这份文档。

表 5.24. 外部命令缓冲队列数

格式：	**external_command_buffer_slots=<#>**
样例：	**external_command_buffer_slots=512**

注 意：这是个高级特性。该选项决定了 Nagios 将使用多少缓冲队列来缓存外部命令，外部命令是从一个工作线程从外部命令文件将命令读入的，但这些外部命令 还没有被 Nagios 的主守护程序处理。缓冲中的每个位置可以处理一个外部命令，所以这个选项决定了有多少命令可以被缓冲处理。为了对一个有大量强制检测 系统(比如分布式系统安装)进行安装时，你可能需要降低这个值。你要考虑

使用 MRTG 工具来绘制外部命令缓冲的利用率图表，如何配置绘制图表可阅读这篇文档。

表 5.25. 互锁文件

格式：	**lock_file=<file_name>**
样例：	**lock_file=/tmp/nagios.lock**

该选项指定了 Nagios 在以守护态运行(以-d 命令行参数运行)时在哪个位置上创建互锁文件。该文件包含有运行 Nagios 的进程 id 值(PID)。

表 5.26. 状态保持选项

格式：	**retain_state_information=<0/1>**
样例：	**retain_state_information=1**

该选项决定了 Nagios 是否要在程序的两次启动之间保存主机和服务的状态信息。如果你使能了这个选项，你应预先给出了 state_retention_file 变量的值，当选项使能时，Nagios 将会在程序停止(或重启)时保存全部的主机和服务的状态信息并且会在启动时再次预读入保存的状态信息。

1. 0 = 不保存状态保持信息
2. 1 = 保留状态保持信息(默认)

表 5.27. 状态保持文件

格式：	**state_retention_file=<file_name>**
样例：	**state_retention_file=/usr/local/nagios/var/retention.dat**

该文件用于在 Nagios 停止之前保存状态、停机时间和注释等信息。当 Nagios 重启时它会在开始监控工作之前使用保存于这个文件里的信息用于初始化主机与服务的状态。为使 Nagios 在程序的启动之间利用状态保持信息，你必须使能 retain_state_information 选项。

表 5.28. 自动状态保持的更新间隔

格式：	**retention_update_interval=<minutes>**
样例：	**retention_update_interval=60**

该选项决定了 Nagios 需要以什么频度(分钟为单位)在正常操作时自动地保存状态保持信息。如果你把这个值设置为 0，Nagios 将不会以规则的间隔保存状态保

持数据，但是 Nagios 仍旧会在停机或重启之前做保存状态保持数据的工作。如果你关闭了状态保持功能(用 retain_state_information 选项设置)，这个选项值将无效。

表 5.29. 程序所用状态的使用选项

格式：	**use_retained_program_state=<0/1>**
样例：	**use_retained_program_state=1**

这个设置将决定了 Nagios 是否要使用保存于状态保持文件之中的值以更新程序范围内的变量状态。有些程序范围内的变量的状态将在程序重启时被保存于状态保持文件之中，包括 enable_notifications、enable_flap_detection、enable_event_handlers、execute_service_checks 和 accept_passive_service_checks 选项。如果你没有使用 retain_state_information 状态保持选项使能，这个选项将无效。

1. 0 = 不使用程序变量的状态值
2. 1 = 使用状态保持文件中的程序变量状态记录(默认)

表 5.30. 使用保持计划表信息选项

格式：	**use_retained_scheduling_info=<0/1>**
样例：	**use_retained_scheduling_info=1**

该选项决定 Nagios 在重启时是否要使用主机和服务的保持计划表信息(下次检测时间)。如果增加了很多数量(或很大百分比)的主机和服务，建议你在首次重启动 Nagios 时关闭选项，因为这个选项将会使初始检测误入歧途。其他情况下你可以要使能这个选项。

1. 0 = 不使用计划表信息
2. 1 = 使用保存的计划表信息(默认)

表 5.31. 保持主机和服务属性掩码

格式：	retained_host_attribute_mask=<number> retained_service_attribute_mask=<number>
样例：	retained_host_attribute_mask=0 retained_service_attribute_mask=0

警告：这是个高级特性。你需要读一下源程序以看清楚它是如何起效果的。

该选项决定了哪个主机和服务的属性在程序重启时不会被保留。这些选项值是与指定的"MODATTR_"值进行*按位与*运算出的，MODATTR_在源程序的include/common.h 里定义，默认情况下，全部主机和服务的属性都会被保持。

表 5.32. 保持进程属性掩码

格式：	retained_process_host_attribute_mask=<number> retained_process_service_attribute_mask=<number>
样例：	retained_process_host_attribute_mask=0 retained_process_service_attribute_mask=0

警告：这是个高级特性。你需要读一下源程序以看清楚它是如何起效果的。

该选项决定了哪个进程属性在程序重启时不会被保留。有两个属性掩码因为经常是主机和服务的进程属性可以分别被修改。例如，主机检测在程序层面上被关闭，而服务检测仍旧被打开。这些选项值是与指定的"MODATTR_"值进行*按位与*运算出的，MODATTR_在源程序的 include/common.h 里定义，默认情况下，全部主机和服务的属性都会被保持。

表 5.33. 保持联系人属性掩码

格式：	retained_contact_host_attribute_mask=<number> retained_contact_service_attribute_mask=<number>
样例：	retained_contact_host_attribute_mask=0 retained_contact_service_attribute_mask=0

警告：这是个高级特性。你需要读一下源程序以看清楚它是如何起效果的。

该选项决定了哪个联系人属性在程序重启时不会被保留。有两个属性掩码因为经常是主机和服务的联系人属性可以分别被修改。这些选项值是与指定的"MODATTR_"值进行*按位与*运算出的，MODATTR_在源程序的 include/common.h 里定义，默认情况下，全部主机和服务的属性都会被保持。

表 5.34. Syslog 日志选项

格式：	**use_syslog=<0/1>**
样例：	**use_syslog=1**

该选项决定了是否将日志信息记录到本地的 Syslog 中。可用的值有：

1. 0 = 不使用 Syslog 机制
2. 1 = 使用 Syslog 机制

表 5.35. 通知记录日志选项

格式：	**log_notifications=<0/1>**
样例：	**log_notifications=1**

该选项决定了是否将通知信息记录进行记录，如果有很多联系人或是有规律性的服务故障时，记录文件将会增长很快。使用这个选项来保存已发出的通知记录。

1. 0 = 不记录通知
2. 1 = 记录通知

表 5.36. 服务检测重试记录选项

格式：	**log_service_retries=<0/1>**
样例：	**log_service_retries=1**

该选项决定了是否将服务检测重试进行记录。服务检测重试发生在服务检测结果返回一个异常状态信息之时，而且你已经配置 Nagios 在对故障出现时进行一次以上的服务检测重试。此时有服务状态被认为是处理“软”故障状态。当调试 Nagios 或对服务的事件处理进行测试时记录下服务检测的重试是非常有用的。

1. 0 = 不记录服务检测重试
2. 1 = 记录服务检测重试

表 5.37. 主机检测重试记录选项

格式：	**log_host_retries=<0/1>**
样例：	**log_host_retries=1**

该选项决定了是否将主机检测重试进行记录。当调试 Nagios 或对主机的事件处理进行测试时记录下主机检测的重试是非常有用的。

1. 0 = 不记录主机检测重试
2. 1 = 记录主机检测重试

表 5.38. 事件处理记录选项

格式：	**log_event_handlers=<0/1>**
样例：	**log_event_handlers=1**

该选项决定了是否将服务和主机的事件处理进行记录。一旦发生服务或主机状态迁移时，可选的事件处理命令会被执行。当调试 Nagios 或首次尝试事件处理脚本时记录下事件处理是非常有用的。

1. 0 = 不记录事件处理
2. 1 = 记录事件处理

表 5.39. 初始状态记录选项

格式：	**log_initial_states=<0/1>**
样例：	**log_initial_states=1**

该选项决定了 Nagios 是否要强行记录全部的主机和服务的初始状态，即便状态报告是 OK 也要记录。只是在第一次检测发现主机和服务有异常时才会记录下初始状态。如果想用应用程序扫描一段时间内的主机和服务状态以生成统计报告时，使能这个选项将有很有帮助。

1. 0 = 不记录初始状态(默认)
2. 1 = 记录初始状态

表 5.40. 外部命令记录选项

格式：	**log_external_commands=<0/1>**
样例：	**log_external_commands=1**

该选项决定了 Nagios 是否要记录外部命令，外部命令是从 command_file 外部命令文件中提取的。注意：这个选项并不控制是否要对强制服务检测（一种外部命令类型)进行记录。为使能或关闭对强制服务检测的记录，使用 log_passive_checks 强制检测记录选项。

1. 0 = 不记录外部命令
2. 1 = 记录外部命令(默认)

表 5.41. 强制检测记录选项

格式：	**log_passive_checks=<0/1>**
样例：	**log_passive_checks=1**

该选项决定了Nagios是否要记录来自于command_file外部命令文件的强制主机和强制服务检测命令。如果要设置一个分布式监控环境或是计划在规整的基础上要对大量的强制检测的结果进行处理时，需要关闭这个选项以防止日志文件过份增长。

1. 0 = 不记录强制检测
2. 1 = 记录强制检测(默认)

表 5.42. 全局主机事件处理选项

格式：	**global_host_event_handler=<command>**
样例：	**global_host_event_handler=log-host-event-to-db**

该选项指定了当每个主机状态迁移时需要执行的主机事件处理命令。全局事件处理命令将优于在每个主机定义的事件处理命令而立即执行。**命令**参数是在对象配置文件里定义的命令的短名称。由 event_handler_timeout 事件处理超时选项控制的这个命令可运行的最大次数。更多的有关事件处理的信息可以查阅这篇文档。

表 5.43. 全局服务事件处理选项

格式：	**global_service_event_handler=<command>**
样例：	**global_service_event_handler=log-service-event-to-db**

该选项指定了当每个服务状态迁移时需要执行的服务事件处理命令。全局事件处理命令将优于在每个服务定义的事件处理命令而立即执行。**命令**参数是在对象配置文件里定义的命令的短名称。由 event_handler_timeout 事件处理超时选项控制的这个命令可运行的最大次数。更多的有关事件处理的信息可以查阅这篇文档。

表 5.44. 检测休止时间间隔

格式：	**sleep_time=<seconds>**
样例：	**sleep_time=1**

它指定了 Nagios 在进行计划表的下一次服务或主机检测命令执行之前应该休止多少秒。注意 Nagios 只是在已经进行了服务故障的排队检测之后才会休止。

表 5.45. 服务检测迟滞间隔计数方法

格式：	**service_inter_check_delay_method=<n/d/s/x.xx>**
样例：	**service_inter_check_delay_method=s**

该 选项容许你控制服务检测将如何初始展开事件队列。 Using a "smart" delay calculation (the default) will cause Nagios to calculate an average check interval and spread initial checks of all services out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally not recommended, as it will cause all service checks to be scheduled for execution at the same time. This means that you will generally have large CPU spikes when the services are all executed in parallel. More information on how to estimate how the inter-check delay affects service check scheduling can be found here. Values are as follows:

1. n = Don't use any delay - schedule all service checks to run immediately (i.e. at the same time!)
2. d = Use a "dumb" delay of 1 second between service checks
3. s = Use a "smart" delay calculation to spread service checks out evenly (default)
4. x.xx = Use a user-supplied inter-check delay of x.xx seconds

表 5.46. 最大服务检测传播时间

格式：	**max_service_check_spread=<minutes>**
样例：	**max_service_check_spread=30**

This option determines the maximum number of minutes from when Nagios starts that all services (that are scheduled to be regularly checked) are checked. This option will automatically adjust the service_inter_check_delay_methodservice inter-check delay method (if necessary) to ensure that the initial checks of all services occur within the timeframe you specify. In general, this option will not have an affect on service check scheduling if scheduling information is being retained using the use_retained_scheduling_infouse_retained_scheduling_info option. 默认值是 30 分钟。

表 5.47. 服务交错因子

格式：	**service_interleave_factor=<s|x>**
样例：	**service_interleave_factor=s**

This variable determines how service checks are interleaved. Interleaving allows for a more even distribution of service checks, reduced load on remote hosts, and faster overall detection of host problems. Setting this value to 1 is equivalent to not interleaving the service checks (this is how versions of Nagios previous to 0.0.5 worked). Set this value to s (smart) for automatic calculation of the interleave factor unless you have a specific reason to change it. The best way to understand how interleaving

works is to watch the status CGI (detailed view) when Nagios is just starting. You should see that the service check results are spread out as they begin to appear. More information on how interleaving works can be found here.

1. **x** = A number greater than or equal to 1 that specifies the interleave factor to use. An interleave factor of 1 is equivalent to not interleaving the service checks.
2. s = Use a "smart" interleave factor calculation (default)

表 5.48. 最大并发服务检测数

格式：	**max_concurrent_checks=<max_checks>**
样例：	**max_concurrent_checks=20**

该 选项可指定在任意给定时间里可被同时运行的服务检测命令的最大数量。如果指定这个值为 1，则说明不允许任何并行服务检测，如果指定为 0(默认值)则是对并 行服务检测。你须按照可运行 Nagios 的机器上的机器资源情况修改这个值，因为它会直接影响系统最大负荷，它施加于系统(处理器利用率、内存使用率等) 之上。更多的关于如何评估需要设置多少并行检测值的信息可以查阅这篇文档。

表 5.49. 检测结果的回收频度

格式：	**check_result_reaper_frequency=<frequency_in_seconds>**
样例：	**check_result_reaper_frequency=5**

该选项控制检测结果的回收事件的处理频度(**以秒为单位**)。从主机和服务的检测过程里“回收”事件处理结果将是对已经执行结束的检测。事件的构成在 Nagios 里是监控逻辑里的核心内容。

表 5.50. 最大检测结果回收时间段

格式：	**max_check_result_reaper_time=<seconds>**
样例：	**max_check_result_reaper_time=30**

该选项决定主机和服务检测结果回收时对结果回收时间段的控制，这个值是个**以秒为单位**的 最大时间跨度。从主机和服务的检测过程里“回收”事件处理结果将是对已经执行结束的检测。如果有许多结果要处理，回收事件过程将占用很长时间来完成它，这 将延迟对新的主机和服务检测的执行。该选项可以限制从检测结果得到与回收处理之间的最大时间间隔以使 Nagios 可以完成对其他监控逻辑的转换处理。

表 5.51. 检测结果保存路径

格式：	**check_result_path=<path>**
样例：	**check_result_path=/var/spool/nagios/checkresults**

该选项决定了 Nagios 将在处理检测结果之前使用哪个目录来保存主机和服务检测结果。这个目录不能保存其他文件，因为 Nagios 会周期性地清理这个目录下的旧文件(更多信息见 max_check_result_file_age 选项)。

注意:确保只有一个 Nagios 的实例在操作检测结果保存路径。如果有多个 Nagios 的实例来操作相同的目录，将会因为错误的 Nagios 实例不正确地处理导致有错误结果!

表 5.52. 检测结果文件的最大生存时间

格式：	**max_check_result_file_age=<seconds>**
样例：	**max_check_result_file_age=3600**

该选项决定用最大多少秒来限定那些在 check_result_path 设置所指向目录里的检测结果文件是合法的。如果检测结果文件超出了这个门限，Nagios 将会把过旧的文件删除而且不会处理内含的检测结果。若设置该选项为 0，Nagios 将处理全部的检测结果文件一即便这些文件比你的硬件还老旧。

表 5.53. 主机检测迟滞间隔计数方式

格式：	**host_inter_check_delay_method=<n/d/s/x.xx>**
样例：	**host_inter_check_delay_method=s**

This option allows you to control how host checks that are scheduled to be checked on a regular basis are initially "spread out" in the event queue. Using a "smart" delay calculation (the default) will cause Nagios to calculate an average check interval and spread initial checks of all hosts out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally not recommended. Using no delay will cause all host checks to be scheduled for execution at the same time. More information on how to estimate how the inter-check delay affects host check scheduling can be found here.Values are as follows:

1. n = Don't use any delay - schedule all host checks to run immediately (i.e. at the same time!)
2. d = Use a "dumb" delay of 1 second between host checks
3. s = Use a "smart" delay calculation to spread host checks out evenly (default)

4. x.xx = Use a user-supplied inter-check delay of x.xx seconds

表 5.54. 最大主机检测传播时间

格式：	**max_host_check_spread=<minutes>**
样例：	**max_host_check_spread=30**

This option determines the maximum number of minutes from when Nagios starts that all hosts (that are scheduled to be regularly checked) are checked. This option will automatically adjust the host_inter_check_delay_methodhost inter-check delay method (if necessary) to ensure that the initial checks of all hosts occur within the timeframe you specify. In general, this option will not have an affect on host check scheduling if scheduling information is being retained using the use_retained_scheduling_infouse_retained_scheduling_info option. Default value is 30 (minutes).

表 5.55. 计数间隔长度

格式：	**interval_length=<seconds>**
样例：	**interval_length=60**

该选项指定了“单位间隔”是多少秒数，单位间隔用于计数计划队列处理、再次通知等。单位间隔在对象配置文件被用于决定以何频度运行服务检测、以何频度与联系人再通知等。

重要：默认值是 60，这说明在对象配置文件里设定的“单位间隔”是 60 秒(1 分钟)。我没测试过其他值，所以如果要用其他值要自担风险！

表 5.56. 自动计划检测选项

格式：	**auto_reschedule_checks=<0/1>**
样例：	**auto_reschedule_checks=1**

该选项决定了 Nagios 是否要试图自动地进行计划的自主检测主机与服务以使在之后的时间里检测更为“平滑”。这可以使得监控主机保持一个均衡的负载，也使得在持续检测之间的保持相对一致，其代价是要更刚性地按计划执行检测工作。

WARNING: THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THIS OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

表 5.57. Auto-Rescheduling Interval

格式：	**auto_rescheduling_interval=<seconds>**
样例：	**auto_rescheduling_interval=30**

This option determines how often (in seconds) Nagios will attempt to automatically reschedule checks. This option only has an effect if the auto_reschedule_checksauto_reschedule_checks option is enabled. Default is 30 seconds.

WARNING: THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THE AUTO-RESCHEDULING OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

表 5.58. Auto-Rescheduling Window

格式：	**auto_rescheduling_window=<seconds>**
样例：	**auto_rescheduling_window=180**

This option determines the "window" of time (in seconds) that Nagios will look at when automatically rescheduling checks. Only host and service checks that occur in the next X seconds (determined by this variable) will be rescheduled. This option only has an effect if the auto_reschedule_checksauto_reschedule_checks option is enabled. Default is 180 seconds (3 minutes).

WARNING: THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THE AUTO-RESCHEDULING OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

表 5.59. 进取式主机检测选项

格式：	**use_aggressive_host_checking=<0/1>**
样例：	**use_aggressive_host_checking=0**

Nagios tries to be smart about how and when it checks the status of hosts. In general, disabling this option will allow Nagios to make some smarter decisions and check hosts a bit faster. Enabling this option will increase the amount of time required to check hosts, but may improve reliability a bit. Unless you have problems with Nagios not recognizing that a host recovered, I would suggest not enabling this option.

1. 0 = Don't use aggressive host checking (default)
2. 1 = Use aggressive host checking

表 5.60. 传递强制主机检测结果选项

格式：	**translate_passive_host_checks=<0/1>**
样例：	**translate_passive_host_checks=1**

This option determines whether or not Nagios will DOWN/UNREACHABLE passive host check results to their "correct" state from the viewpoint of the local Nagios instance. This can be very useful in distributed and failover monitoring installations. More information on passive check state translation can be found here.

1. 0 = Disable check translation (default)
2. 1 = Enable check translation

表 5.61. Passive Host Checks Are SOFT Option

格式：	**passive_host_checks_are_soft=<0/1>**
样例：	**passive_host_checks_are_soft=1**

This option determines whether or not Nagios will treat passive host checks as HARD states or SOFT states. By default, a passive host check result will put a host into a HARD state type. You can change this behavior by enabling this option.

1. 0 = Passive host checks are HARD (default)
2. 1 = Passive host checks are SOFT

表 5.62. Predictive Host Dependency Checks Option

格式：	**enable_predictive_host_dependency_checks=<0/1>**
样例：	**enable_predictive_host_dependency_checks=1**

This option determines whether or not Nagios will execute predictive checks of hosts that are being dependended upon (as defined in host dependencies) for a particular host when it changes state.

Predictive checks help ensure that the dependency logic is as accurate as possible. More information on how predictive checks work can be found here.

1. 0 = Disable predictive checks
2. 1 = Enable predictive checks (default)

表 5.63. Predictive Service Dependency Checks Option

格式：	**enable_predictive_service_dependency_checks=<0/1>**
样例：	**enable_predictive_service_dependency_checks=1**

This option determines whether or not Nagios will execute predictive checks of services that are being dependended upon (as defined in service dependencies) for a particular service when it changes state.

Predictive checks help ensure that the dependency logic is as accurate as possible. More information on how predictive checks work can be found here.

1. 0 = Disable predictive checks
2. 1 = Enable predictive checks (default)

表 5.64. Cached Host Check Horizon

格式：	**cached_host_check_horizon=<seconds>**
样例：	**cached_host_check_horizon=15**

This option determines the maximum amount of time (in seconds) that the state of a previous host check is considered current. Cached host states (from host checks that were performed more recently than the time specified by this value) can improve host check performance immensely. Too high of a value for this option may result in (temporarily) inaccurate host states, while a low value may result in a performance hit for host checks. Use a value of 0 if you want to disable host check caching. More information on cached checks can be found here.

表 5.65. Cached Service Check Horizon

格式：	**cached_service_check_horizon=<seconds>**
样例：	**cached_service_check_horizon=15**

This option determines the maximum amount of time (in seconds) that the state of a previous service check is considered current. Cached service states (from service checks that were performed more recently than the time specified by this value) can improve service check performance when

a lot of service dependencies are used. Too high of a value for this option may result in inaccuracies in the service dependency logic. Use a value of 0 if you want to disable service check caching. More information on cached checks can be found here.

表 5.66. Large Installation Tweaks Option

格式：	**use_large_installation_tweaks=<0/1>**
样例：	**use_large_installation_tweaks=0**

This option determines whether or not the Nagios daemon will take several shortcuts to improve performance. These shortcuts result in the loss of a few features, but larger installations will likely see a lot of benefit from doing so. More information on what optimizations are taken when you enable this option can be found here.

1. 0 = Don't use tweaks (default)
2. 1 = Use tweaks

表 5.67. 子进程内存选项

格式：	**free_child_process_memory=<0/1>**
样例：	**free_child_process_memory=0**

This option determines whether or not Nagios will free memory in child processes when they are fork()ed off from the main process. By default, Nagios frees memory. However, if the use_large_installation_tweaks option is enabled, it will not. By defining this option in your configuration file, you are able to override things to get the behavior you want.

1. 0 = Don't free memory
2. 1 = Free memory

表 5.68. 子进程二次派生选项

格式：	**child_processes_fork_twice=<0/1>**
样例：	**child_processes_fork_twice=0**

This option determines whether or not Nagios will fork() child processes twice when it executes host and service checks. By default, Nagios fork()s twice. However, if the use_large_installation_tweaks option is enabled,

it will only fork() once. By defining this option in your configuration file, you are able to override things to get the behavior you want.

1. 0 = Fork() just once
2. 1 = Fork() twice

表 5.69. 环境变量中标准宏可用性选项

格式：	**enable_environment_macros=<0/1>**
样例：	**enable_environment_macros=0**

This option determines whether or not the Nagios daemon will make all standard macros available as environment variables to your check, notification, event hander, etc. commands. In large Nagios installations this can be problematic because it takes additional memory and (more importantly) CPU to compute the values of all macros and make them available to the environment.

1. 0 = Don't make macros available as environment variables
2. 1 = Make macros available as environment variables (default)

表 5.70. Flap Detection Option

格式：	**enable_flap_detection=<0/1>**
样例：	**enable_flap_detection=0**

This option determines whether or not Nagios will try and detect hosts and services that are "flapping". Flapping occurs when a host or service changes between states too frequently, resulting in a barrage of notifications being sent out. When Nagios detects that a host or service is flapping, it will temporarily suppress notifications for that host/service until it stops flapping. Flap detection is very experimental at this point, so use this feature with caution! More information on how flap detection and handling works can be found here.注意：如果你使能 retain_state_information 状态保持选项(保存于 state_retention_file 状态保持文件中)而忽略这个设置，**除非**你已经关闭 use_retained_program_state 选项。如果你想在保持选项使能(且 use_retained_program_state 选项使能)的情况下修改这个选项，你只得用适当的外部命令或是通过 Web 接口来修改它。选项可用的值有：

1. 0 = Don't enable flap detection (default)
2. 1 = Enable flap detection

表 5.71. Low Service Flap Threshold

格式：	**low_service_flap_threshold=<percent>**
样例：	**low_service_flap_threshold=25.0**

This option is used to set the low threshold for detection of service flapping. For more information on how flap detection and handling works (and how this option affects things) read this.

表 5.72. High Service Flap Threshold

格式：	**high_service_flap_threshold=<percent>**
样例：	**high_service_flap_threshold=50.0**

This option is used to set the low threshold for detection of service flapping. For more information on how flap detection and handling works (and how this option affects things) read this.

表 5.73. Low Host Flap Threshold

格式：	**low_host_flap_threshold=<percent>**
样例：	**low_host_flap_threshold=25.0**

This option is used to set the low threshold for detection of host flapping. For more information on how flap detection and handling works (and how this option affects things) read this.

表 5.74. High Host Flap Threshold

格式：	**high_host_flap_threshold=<percent>**
样例：	**high_host_flap_threshold=50.0**

This option is used to set the low threshold for detection of host flapping. For more information on how flap detection and handling works (and how this option affects things) read this.

表 5.75. Soft State Dependencies Option

格式：	**soft_state_dependencies=<0/1>**
样例：	**soft_state_dependencies=0**

This option determines whether or not Nagios will use soft state information when checking host and service dependencies. Normally Nagios will only use the latest hard host or service state when checking dependencies. If you want it to use the latest state (regardless of whether its a soft or hard state type), enable this option.

1. 0 = Don't use soft state dependencies (default)
2. 1 = Use soft state dependencies

表 5.76. 服务检测超时

格式：	**service_check_timeout=<seconds>**
样例：	**service_check_timeout=60**

This is the maximum number of seconds that Nagios will allow service checks to run. If checks exceed this limit, they are killed and a 紧急 state is returned. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each service check normally finishes executing within this time limit. If a service check runs longer than this limit, Nagios will kill it off thinking it is a runaway processes.

表 5.77. 主机检测超时

格式：	**host_check_timeout=<seconds>**
样例：	**host_check_timeout=60**

This is the maximum number of seconds that Nagios will allow host checks to run. If checks exceed this limit, they are killed and a 紧急 state is returned and the host will be assumed to be DOWN. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each host check normally finishes executing within this time limit. If a host check runs longer than this limit, Nagios will kill it off thinking it is a runaway processes.

表 5.78. 事件处理超时

格式：	**event_handler_timeout=<seconds>**
样例：	**event_handler_timeout=60**

This is the maximum number of seconds that Nagios will allow event handlers to be run. If an event handler exceeds this time limit it will be killed and a warning will be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off commands which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each event handler command normally finishes executing within this time limit. If an event handler runs longer than this limit, Nagios will kill it off thinking it is a runaway processes.

表 5.79. 通知超时

格式：	**notification_timeout=<seconds>**
样例：	**notification_timeout=60**

This is the maximum number of seconds that Nagios will allow notification commands to be run. If a notification command exceeds this time limit it will be killed and a warning will be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off commands which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each notification command finishes executing within this time limit. If a notification command runs longer than this limit, Nagios will kill it off thinking it is a runaway processes.

表 5.80. Obsessive Compulsive Service Processor Timeout

格式：	**ocsp_timeout=<seconds>**
样例：	**ocsp_timeout=5**

This is the maximum number of seconds that Nagios will allow an ocsp_commandobsessive compulsive service processor command to be run. If

a command exceeds this time limit it will be killed and a warning will be logged.

表 5.81. Obsessive Compulsive Host Processor Timeout

格式：	**ochp_timeout=<seconds>**
样例：	**ochp_timeout=5**

This is the maximum number of seconds that Nagios will allow an ochp_commandobsessive compulsive host processor command to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

表 5.82. 性能数据处理命令超时

格式：	**perfdata_timeout=<seconds>**
样例：	**perfdata_timeout=5**

This is the maximum number of seconds that Nagios will allow a host_perfdata_commandhost performance data processor command or service_perfdata_commandservice performance data processor command to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

表 5.83. Obsess Over Services Option

格式：	**obsess_over_services=<0/1>**
样例：	**obsess_over_services=1**

This value determines whether or not Nagios will "obsess" over service checks results and run the ocsp_commandobsessive compulsive service processor command you define. I know - funny name, but it was all I could think of. This option is useful for performing distributed monitoring. If you're not doing distributed monitoring, don't enable this option.

1. 0 = Don't obsess over services (default)
2. 1 = Obsess over services

表 5.84. Obsessive Compulsive Service Processor Command

格式：	**ocsp_command=<command>**
样例：	**ocsp_command=obsessive_service_handler**

This option allows you to specify a command to be run after **every** service check, which can be useful in distributed monitoring. This command is executed after any event handler or notification commands. The **command** argument is the short name of a command definition that you define in your 对象配置文件. The maximum amount of time that this command can run is controlled by the ocsp_timeoutocsp_timeout option. More information on distributed monitoring can be found here. This command is only executed if the obsess_over_servicesobsess_over_services option is enabled globally and if the **obsess_over_service** directive in the service definition is enabled.

表 5.85. Obsess Over Hosts Option

格式：	**obsess_over_hosts=<0/1>**
样例：	**obsess_over_hosts=1**

This value determines whether or not Nagios will "obsess" over host checks results and run the ochp_commandobsessive compulsive host processor command you define. I know - funny name, but it was all I could think of. This option is useful for performing distributed monitoring. If you're not doing distributed monitoring, don't enable this option.

1. 0 = Don't obsess over hosts (default)
2. 1 = Obsess over hosts

表 5.86. Obsessive Compulsive Host Processor Command

格式：	**ochp_command=<command>**
样例：	**ochp_command=obsessive_host_handler**

This option allows you to specify a command to be run after **every** host check, which can be useful in distributed monitoring. This command is executed after any event handler or notification commands. The **command** argument is the short name of a command definition that you define in your 对象配置文件. The maximum amount of time that this command can run is controlled by the ochp_timeoutochp_timeout option. More information on distributed monitoring can be found here. This command is only executed if the obsess_over_hostsobsess_over_hosts option is enabled globally and if the **obsess_over_host** directive in the host definition is enabled.

表 5.87. 性能数据处理选项

格式：	**process_performance_data=<0/1>**
样例：	**process_performance_data=1**

该选项决定 Nagios 是否要处理主机和服务检测性能数据。

1. 0 = Don't process performance data (default)
2. 1 = Process performance data

表 5.88. 主机性能数据处理命令

格式：	**host_perfdata_command=<command>**
样例：	**host_perfdata_command=process-host-perfdata**

This option allows you to specify a command to be run after **every** host check to process host performance data that may be returned from the check. The **command** argument is the short name of a command definition that you define in your 对象配置文件. This command is only executed if the process_performance_dataprocess_performance_data option is enabled globally and if the **process_perf_data** directive in the host definition is enabled.

表 5.89. 服务性能数据处理命令

格式：	**service_perfdata_command=<command>**
样例：	**service_perfdata_command=process-service-perfdata**

This option allows you to specify a command to be run after **every** service check to process service performance data that may be returned from the check. The **command** argument is the short name of a command definition that you define in your 对象配置文件. This command is only executed if the process_performance_dataprocess_performance_data option is enabled globally and if the **process_perf_data** directive in the service definition is enabled.

表 5.90. 主机性能数据文件

格式：	**host_perfdata_file=<file_name>**
样例：	**host_perfdata_file=/usr/local/nagios/var/host-perfdata.dat**

This option allows you to specify a file to which host performance data will be written after every host check. Data will be written to the performance file as specified by the host_perfdata_file_templatehost_perfdata_file_template option. Performance data is only written to this file if the process_performance_dataprocess_performance_data option is enabled globally and if the process_perf_data directive in the host definition is enabled.

表 5.91. 服务性能数据文件

格式：	**service_perfdata_file=<file_name>**
样例：	**service_perfdata_file=/usr/local/nagios/var/service-perfdata.dat**

This option allows you to specify a file to which service performance data will be written after every service check. Data will be written to the performance file as specified by the service_perfdata_file_template option. Performance data is only written to this file if the process_performance_dataprocess_performance_data option is enabled globally and if the process_perf_data directive in the service definition is enabled.

表 5.92. 主机性能数据文件模板

格式：	**host_perfdata_file_template=<template>**
样例：	host_perfdata_file_template=[HOSTPERFDATA]\t$TIMET$\t$HOSTNAME$\t$HOSTEXECUTIONTIME$ \t$HOSTOUTPUT$\t$HOSTPERFDATA$

This option determines what (and how) data is written to the host_perfdata_filehost performance data file. The template may contain macros, special characters (\t for tab, \r for carriage return, \n for newline) and plain text. A newline is automatically added after each write to the performance data file.

表 5.93. 服务性能数据文件模板

格式：	**service_perfdata_file_template=<template>**
样例：	service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\t$HOSTN

	AME$\t$SERVICEDESC$\t $SERVICEEXECUTIONTIME$\t$SERVICELATENCY$\t$SERVICEOUTPUT$\t$SERVICEPERFDATA$

This option determines what (and how) data is written to the service performance data file. The template may contain macros, special characters (\t for tab, \r for carriage return, \n for newline) and plain text. A newline is automatically added after each write to the performance data file.

表 5.94. 主机性能数据文件打开方式

格式：	**host_perfdata_file_mode=<mode>**
样例：	**host_perfdata_file_mode=a**

This option determines how the host_perfdata_filehost performance data file is opened. Unless the file is a named pipe you'll probably want to use the default mode of append.

1. a = Open file in append mode (default)
2. w = Open file in write mode
3. p = Open in non-blocking read/write mode (useful when writing to pipes)

表 5.95. 性能数据文件打开方式

格式：	**service_perfdata_file_mode=<mode>**
样例：	**service_perfdata_file_mode=a**

This option determines how the service performance data file is opened. Unless the file is a named pipe you'll probably want to use the default mode of append.

1. a = Open file in append mode (default)
2. w = Open file in write mode
3. p = Open in non-blocking read/write mode (useful when writing to pipes)

表 5.96. 主机性能数据文件处理间隔

格式：	**host_perfdata_file_processing_interval=<seconds>**
样例：	**host_perfdata_file_processing_interval=0**

This option allows you to specify the interval (in seconds) at which the host_perfdata_filehost performance data file is processed using the host_perfdata_file_processing_commandhost performance data file processing command. A value of 0 indicates that the performance data file should not be processed at regular intervals.

表 5.97. 服务性能数据文件处理间隔

格式：	**service_perfdata_file_processing_interval=<seconds>**
样例：	**service_perfdata_file_processing_interval=0**

This option allows you to specify the interval (in seconds) at which the service_perfdata_fileservice performance data file is processed using the service_perfdata_file_processing_commandservice performance data file processing command. A value of 0 indicates that the performance data file should not be processed at regular intervals.

表 5.98. 主机性能数据文件处理命令

格式：	**host_perfdata_file_processing_command=<command>**
样例：	**host_perfdata_file_processing_command=process-host-perfdata-file**

This option allows you to specify the command that should be executed to process the host_perfdata_filehost performance data file. The command argument is the short name of a command definition that you define in your 对象配置文件. The interval at which this command is executed is determined by the host_perfdata_file_processing_intervalhost_perfdata_file_processing_interval directive.

表 5.99. 服务性能数据文件处理命令

格式：	**service_perfdata_file_processing_command=<command>**
样例：	**service_perfdata_file_processing_command=process-service-perfdata-file**

This option allows you to specify the command that should be executed to process the service_perfdata_fileservice performance data file. The command argument is the short name of a command definition that you define in your 对象配置文件. The interval at which this command is executed is determined by the service_perfdata_file_processing_intervalservice_perfdata_file_processing_interval directive.

表 5.100. 孤立服务检测选项

格式：	**check_for_orphaned_services=<0/1>**
样例：	**check_for_orphaned_services=1**

This option allows you to enable or disable checks for orphaned service checks. Orphaned service checks are checks which have been executed and have been removed from the event queue, but have not had any results reported in a long time. Since no results have come back in for the service, it is not rescheduled in the event queue. This can cause service checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a service check. If this option is enabled and Nagios finds that results for a particular service check have not come back, it will log an error message and reschedule the service check. If you start seeing service checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned services.

1. 0 = Don't check for orphaned service checks
2. 1 = Check for orphaned service checks (default)

表 5.101. 孤立主机检测选项

格式：	**check_for_orphaned_hosts=<0/1>**
样例：	**check_for_orphaned_hosts=1**

This option allows you to enable or disable checks for orphaned hoste checks. Orphaned host checks are checks which have been executed and have been removed from the event queue, but have not had any results reported in a long time. Since no results have come back in for the host, it is not rescheduled in the event queue. This can cause host checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a host check. If this option is enabled and Nagios finds that results for a particular host check have not come back, it will log an error message and reschedule the host check. If you start seeing host checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned hosts.

1. 0 = Don't check for orphaned host checks
2. 1 = Check for orphaned host checks (default)

表 5.102. 服务更新检测选项

格式：	**check_service_freshness=<0/1>**
样例：	**check_service_freshness=0**

This option determines whether or not Nagios will periodically check the "freshness" of service checks. Enabling this option is useful for helping to ensure that passive service checks are received in a timely manner. More information on freshness checking can be found here.

1. 0 = Don't check service freshness
2. 1 = Check service freshness (default)

表 5.103. 服务更新检测间隔

格式：	**service_freshness_check_interval=<seconds>**
样例：	**service_freshness_check_interval=60**

This setting determines how often (in seconds) Nagios will periodically check the "freshness" of service check results. If you have disabled service freshness checking (with the check_service_freshnesscheck_service_freshness option), this option has no effect. More information on freshness checking can be found here.

表 5.104. 主机更新检测选项

格式：	**check_host_freshness=<0/1>**
样例：	**check_host_freshness=0**

This option determines whether or not Nagios will periodically check the "freshness" of host checks. Enabling this option is useful for helping to ensure that passive host checks are received in a timely manner. More information on freshness checking can be found here.

1. 0 = Don't check host freshness
2. 1 = Check host freshness (default)

表 5.105. 主机更新检测间隔

格式：	**host_freshness_check_interval=<seconds>**
样例：	**host_freshness_check_interval=60**

This setting determines how often (in seconds) Nagios will periodically check the "freshness" of host check results. If you have disabled host freshness checking (with the check_host_freshnesscheck_host_freshness option), this option has no effect. More information on freshness checking can be found here.

表 5.106. Additional Freshness Threshold Latency Option

格式：	**additional_freshness_latency=<#>**
样例：	**additional_freshness_latency=15**

This option determines the number of seconds Nagios will add to any host or services freshness threshold it automatically calculates (e.g. those not specified explicity by the user). More information on freshness checking can be found here.

表 5.107. Embedded Perl Interpreter Option

格式：	**enable_embedded_perl=<0/1>**
样例：	**enable_embedded_perl=1**

This setting determines whether or not the embedded Perl interpreter is enabled on a program-wide basis. Nagios must be compiled with support for embedded Perl for this option to have an effect. More information on the embedded Perl interpreter can be found here.

表 5.108. Embedded Perl Implicit Use Option

格式：	**use_embedded_perl_implicitly=<0/1>**
样例：	**use_embedded_perl_implicitly=1**

This setting determines whether or not the embedded Perl interpreter should be used for Perl plugins/scripts that do not explicitly enable/disable it. Nagios must be compiled with support for embedded Perl for this option to have an effect. More information on the embedded Perl interpreter and the effect of this setting can be found here.

表 5.109. Date Format

格式：	**date_format=<option>**
样例：	**date_format=us**

This option allows you to specify what kind of date/time format Nagios should use in the web interface and date/time macros. Possible options (along with example output) include:

表 5.110.

选项	输出格式	输出样例
us	MM/DD/YYYY HH:MM:SS	06/30/2002 03:15:00
euro	DD/MM/YYYY HH:MM:SS	30/06/2002 03:15:00
iso8601	YYYY-MM-DD HH:MM:SS	2002-06-30 03:15:00
strict-iso8601	YYYY-MM-DDTHH:MM:SS	2002-06-30T03:15:00

表 5.111. 时区选项

格式：	**use_timezone=<tz>**
样例：	**use_timezone=US/Mountain**

This option allows you to override the default timezone that this instance of Nagios runs in. Useful if you have multiple instances of Nagios that need to run from the same server, but have different local times associated with them. If not specified, Nagios will use the system configured timezone.

Note: If you use this option to specify a custom timezone, you will also need to alter the Apache configuration directives for the CGIs to specify the timezone you want. Example:

<Directory "/usr/local/nagios/sbin/">

SetEnv TZ "US/Mountain"

...

</Directory>

表 5.112. 非法对象名字符

格式：	**illegal_object_name_chars=<chars...>**
样例：	**illegal_object_name_chars=`~!$%^&*"|'<>?,()=**

This option allows you to specify illegal characters that cannot be used in host names, service descriptions, or names of other object types. Nagios will allow you to use most characters in object definitions, but I recommend not using the characters shown in the example above. Doing may give you problems in the web interface, notification commands, etc.

表 5.113. 非法宏输出字符

格式：	**illegal_macro_output_chars=<chars...>**
样例：	**illegal_macro_output_chars=`~$^&"|'<>**

This option allows you to specify illegal characters that should be stripped from macros before being used in notifications, event handlers, and other commands. This DOES NOT affect macros used in service or host check commands. You can choose to not strip out the characters shown in the example above, but I recommend you do not do this. Some of these characters are interpreted by the shell (i.e. the backtick) and can lead to security problems. The following macros are stripped of the characters you specify:

$HOSTOUTPUT$, $HOSTPERFDATA$, $HOSTACKAUTHOR$, $HOSTACKCOMMENT$, $SERVICEOUTPUT$, $SERVICEPERFDATA$, $SERVICEACKAUTHOR$, and $SERVICEACKCOMMENT$

表 5.114. 正则表达式选项

格式：	**use_regexp_matching=<0/1>**
样例：	**use_regexp_matching=0**

This option determines whether or not various directives in your 对象定义 will be processed as regular expressions. More information on how this works can be found here.

1. 0 = Don't use regular expression matching (default)
2. 1 = Use regular expression matching

表 5.115. True Regular Expression Matching Option

格式：	**use_true_regexp_matching=<0/1>**
样例：	**use_true_regexp_matching=0**

If you've enabled regular expression matching of various object directives using the use_regexp_matching option, this option will determine when object directives are treated as regular expressions. If this option is disabled (the default), directives will only be treated as regular expressions if the contain *, ?, +, or \.. If this option is enabled, all appropriate directives will be treated as regular expression - be careful when enabling this! More information on how this works can be found here.

1. 0 = Don't use true regular expression matching (default)
2. 1 = Use true regular expression matching

表 5.116. 管理员 EMail 帐号

格式：	**admin_email=<email_address>**
样例：	**admin_email=root@localhost.localdomain**

This is the email address for the administrator of the local machine (i.e. the one that Nagios is running on). This value can be used in notification commands by using the $ADMINEMAIL$macro.

表 5.117. 管理员 BP 机帐号

格式：	**admin_pager=<pager_number_or_pager_email_gateway>**
样例：	**admin_pager=pageroot@localhost.localdomain**

This is the pager number (or pager email gateway) for the administrator of the local machine (i.e. the one that Nagios is running on). The pager number/address can be used in notification commands by using the $ADMINPAGER$macro.

表 5.118. Event Broker Options

格式：	**event_broker_options=<#>**
样例：	**event_broker_options=-1**

This option controls what (if any) data gets sent to the event broker and, in turn, to any loaded event broker modules. This is an advanced option. When in doubt, either broker nothing (if not using event broker modules) or broker everything (if using event broker modules). Possible values are shown below.

1. 0 = Broker nothing
2. -1 = Broker everything
3. # = See BROKER_* definitions in source code (include/broker.h) for other values that can be OR'ed together

表 5.119. Event Broker Modules

格式：	**broker_module=<modulepath> [moduleargs]**
样例：	broker_module=/usr/local/nagios/bin/ndomod.o cfg_file=/usr/local/nagios/etc/ndomod.cfg

This directive is used to specify an event broker module that should by loaded by Nagios at startup. Use multiple directives if you want to load more than one module. Arguments that should be passed to the module at startup are seperated from the module path by a space.

!!! WARNING !!!

Do NOT overwrite modules while they are being used by Nagios or Nagios will crash in a fiery display of SEGFAULT glory. This is a bug/limitation either in dlopen(), the kernel, and/or the filesystem. And maybe Nagios...

The correct/safe way of updating a module is by using one of these methods:

1. Shutdown Nagios, replace the module file, restart Nagios
2. While Nagios is running... delete the original module file, move the new module file into place, restart Nagios

表 5.120. 调试文件

格式：	**debug_file=<file_name>**
样例：	**debug_file=/usr/local/nagios/var/nagios.debug**

This option determines where Nagios should write debugging information. What (if any) information is written is determined by the debug_level and debug_verbosity options. You can have Nagios automaticaly rotate the debug file when it reaches a certain size by using the max_debug_file_size option.

表 5.121. 调试等级

格式：	**debug_level=<#>**

样例：	**debug_level=24**

该选项决定 Nagios 将往 debug_file 文件里写入什么调试信息。下面值是可以逻辑或关系：

1. -1 = Log everything
2. 0 = Log nothing (default)
3. 1 = Function enter/exit information
4. 2 = Config information
5. 4 = Process information
6. 8 = Scheduled event information
7. 16 = Host/service check information
8. 32 = Notification information
9. 64 = Event broker information

表 5.122. Debug Verbosity

格式：	**debug_verbosity=<#>**
样例：	**debug_verbosity=1**

This option determines how much debugging information Nagios should write to the debug_filedebug_file.

1. 0 = Basic information
2. 1 = More detailed information (default)
3. 2 = Highly detailed information

表 5.123. 调试文件最大长度

格式：	**max_debug_file_size=<#>**
样例：	**max_debug_file_size=1000000**

该选项定义了以字节为单位的 debug_file 调试文件最大长度。如果文件增至大于该值，将会自动被命名为.old 扩展名的文件，如果.old 扩展名已经存在，那么旧.old 文件将被删除。这可以保证在 Nagios 调试时磁盘空间不会过多占用而失控。

5.3. 对象配置概览

5.3.1. 什么是对象？

对象是指所有在监控和通知逻辑中涉及到的元素。对象的类型包括：

1. 服务
2. 服务组
3. 主机
4. 主机组
5. 联系人
6. 联系人组
7. 命令
8. 时间周期
9. 通知扩展
10. 通知和执行依赖关系

更多有关对象和它们之间关系的说明见下面。

5.3.2. 对象在哪里定义？

对象可以在一个配置文件 cfg_file 或是多个由主配置文件对象保存目录 cfg_dir 里配置文件来定义。

提示

当按照快速安装指南进行安装后，几个对象配置文件的样例放在了**/usr/local/nagios/etc/objects/**目录下。可以用这些样例文件来搞清楚对象继承关系并学习如何进行自己的对象定义。

5.3.3. 对象如何定义？

对象可以在一个用柔性化模板样式来定义，模板可使得对 Nagios 的配置管理更为容易，有关如果进行对象定义的基本信息可以查阅这篇文件。

一旦熟悉了如何进行对象定义的基础，需要阅读对象继承以在将来应用中配置更为鲁棒(就是尽量使用对象继承关系啦)。经验丰富的使用者可以在对象定义决窍一文中发掘到一些有关对象定义的高级特性.

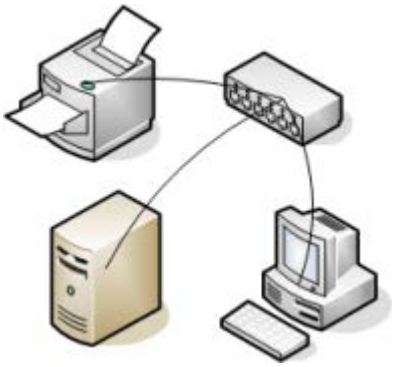
关于对象的解释

下面在一些主要的对象的解释...

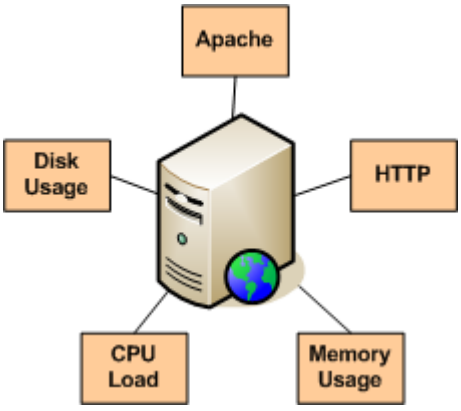
- 主机是监控逻辑中的核心对象之一。主机的重要属性有：
    1. 主机通常在网络中的物理设备(如服务器、工作站、路由器、交换机和打印机等)；
    2. 主机有某种形式的地址(象 IP 或 MAC 地址)；

3. 主机有一个或多个绑定的服务；
4. 主机与其他的主机间可以有父/子节点的关系，通常反应出真实世界里的网络联接关系，而联接关系会在网络可达性逻辑中用到。

- 主机组是一台或多台主机组成的组。主机成组可以如下工作更简单(1)在 Nagios 的 Web 接口里查看相关的主机状态(2)使用对象定义决窍来简化配置。



- 服务监控逻辑中的一个核心对象之一。在主机上的服务用户可以：
  1. 主机的属性(CPU 负荷、磁盘利用率、启动时间等)；
  2. 主机提供的服务(HTTP, POP3, FTP, SSH 等等)；
  3. 其他与主机有关的信息(DNS 记录等)；
- 服务组是一个或多个服务组成的组。服务组可以对如下工作更简单(1)在 Nagios 的 Web 接口里查看相关的服务状态(2)使用对象定义决窍来简化配置。



- 联系人是那些涉及到通知过程中的人：
  1. 有多种通知联系人的方法(对讲机、BP 机、EMail、即时信息等)；
  2. 联系人收到的通知来自于其负责的主机或服务；
- 联系人组是一个或多个联系人组成的组。联系人组可以简化在主机或服务故障时负责的人员划分。

- 时间周期用于控制：
  1. 主机或服务被监控的时间；
  2. 联系人可接收通知的时间；

  时间段时如何工作的信息可以查阅这篇文档。

- 命令是指出 Nagios 用哪个程序、脚本等，它必须可执行后完成：
  1. 主机和服务检测
  2. 通知
  3. 事件处理
  4. 和其他...

5.4. CGI 配置文件选项

注意

当创建或编辑配置文件时，要遵守如下要求：

1. 以符号'#'开头的行将视为注释不做处理；
2. 变量必须是新起的一行 — 变量之前不能有空格符；
3. 变量名是大小写敏感的；

5.4.1. 样例配置文件

提示

一个 CGI 的样例配置文件(/usr/local/nagios/etc/cgi.cfg)已经安装到位，如果你是按照快速安装指南来操作的话。

5.4.2. 配置文件的位置

默认情况下，Nagios 期望的 CGI 配置文件被命名为 cgi.cfg 并且该配置文件被放在了主配置文件指定的位置。如果你想改变名称和位置，你可以在 Apache 里配置一个环境变量叫做 NAGIO_CGI_CONFIG 的(里面设置好文件名和位置)给 CGI 程序用。如何来做可以查看 Apache 文档里的说明。

5.4.3. 配置文件里的变量

下面将给出每个主配置文件里的变量与值选项说明...

表 5.124. 主配置文件的位置

格式：	**main_config_file=<file_name>**
举例：	**main_config_file=/usr/local/nagios/etc/nagios.cfg**

它用于指向主配置文件所在的位置。CGI 模块需要知道在哪里可以得到主配置文件以取得配置信息、当前的主机和服务的状态等。

表 5.125. HTML 文件的系统路径

格式：	**physical_html_path=<path>**
举例：	**physical_html_path=/usr/local/nagios/share**

它用于指明用于服务器或工作站上的 HTML 文件所在的**系统**路径。Nagios 假定文档和图片文件被分别放在了 docs/和 images/两个子目录下。

表 5.126. URL 里的 HTML 路径

格式：	**url_html_path=<path>**
举例：	**url_html_path=/nagios**

如果通过 Web 浏览器来操作 Nagios，你要通过一个 URL 如 http://www.myhost.com/nagios 来操作的话，则需要设置为/nagios。一般是用这个 URL 来操作 Nagios 的 HTML 页面。

表 5.127. 应用认证

格式：	**use_authentication=<0/1>**
举例：	**use_authentication=1**

该选项控制着 CGI 模块里，对于用户操作或是取得信息时是否需要打开认证和授权功能。如果你断定你不使用认证，一定要把 CGI 命令移走以免没有授权的用户发出 Nagios 命令。如果不使用认证功能，CGI 模块不会向 Nagios 发出命令，但我同时也建议你也把 CGI 模块同时移到安全位置。更多的有关设置认证与授权的内容可以查看这个文件。

1. 0 = 不使用认证功能
2. 1 = 使用认主与授权功能(默认值)

表 5.128. 默认用户名

格式：	**default_user_name=<username>**
举例：	**default_user_name=guest**

用这个变量可以设置一个默认的用户来操作 CGI 程序。它可以在一个加密的域里(如在防火墙后建立的 WEB)不需要 WEB 认证就可以操作 CGI 模块。你可能需要这个功能来避免仅仅在一个非加密的服务器上(通过因特网以明文方式来传递你的口令)来做基本的认证。

Important:除非你是在一个加密的 WEB 服务器上并且保证每个进入该域的用户都具备 CGI 操作权，否则的话，你不要定义这个默认用户。如果你决定用它，那么任何一个未经认证的 WEB 服务器用户都可以继承你设定的全部权限！

表 5.129. 系统和进程的信息操作权

格式：	**authorized_for_system_information=<user1>,<user2>,<user3>,...<usern>**
举例：	**authorized_for_system_information=nagiosadmin,theboss**

这是一个以逗号分陋的列表，列举出了在扩展 CGI 信息里查看系统和进程信息的**可认证用户**。在列表中列出的用户并**不会**自动被授权可发出系统和进程的命令。如果你想也同时可以发出系统和进程命令，你必须把这些用户也加到 authorized_for_system_commands 变量之中。更多的如何给 CGI 模块设置认证和配置授权的内容可以查阅这个文档。

表 5.130. 系统和进程的命令操作权

格式：	**authorized_for_system_commands=<user1>,<user2>,<user3>,...<usern>**
举例：	**authorized_for_system_commands=nagiosadmin**

这是一个以逗号分隔的列表，列出了可以通过 CGI 命令发出系统和进程命令的**被认证用户**。在列表中的用户并**没有**被自动授权查看系统和进程的信息。如果你想让用户也同时可以查看系统和进程信息的话，你必须把这些用户也加到 authorized_for_system_information 变量里面。更多的如何给 CGI 模块设置认证和配置授权的内容可以查阅这个文档。

表 5.131. 配置的信息获取权限

格式：	**authorized_for_configuration_information=<user1>,<user2>,<user3>,...<usern>**
举例：	**authorized_for_configuration_information=nagiosadmin**

这是一个以逗号分隔的列表，列出了可以通过配置查看 CGI 里查看配置信息的**可认证用户**。这些列表中的用户可以查看全部的配置好的主机、主机组、服务、联系人、联系人组等的配置信息。更多的如何给 CGI 模块设置认证和配置授权的内容可以查阅这个文档。

表 5.132. 全局主机的信息获取权限

格式：	**authorized_for_all_hosts=<user1>,<user2>,<user3>,...<usern>**
举例：	**authorized_for_all_hosts=nagiosadmin,theboss**

这是一个以逗号分隔的列表，列出了可以查看全部主机的状态和配置信息的**被认证用户**。这些列表中的用户同时被授权查看在全部的服务信息。但列表中的用户并**没有**自动地授权向全部的主机或服务发出命令。如果你想让这些用户同时可以向全部主机和服务发出命令，你必须将用户加入到 authorized_for_all_host_commands 变量里。更多的如何给 CGI 模块设置认证和配置授权的内容可以查阅这个文档。

表 5.133. 全局主机的命令操作权

格式：	**authorized_for_all_host_commands=<user1>,<user2>,<user3>,...<usern>**
举例：	**authorized_for_all_host_commands=nagiosadmin**

这是一个以逗号分隔的列表，列出了可以通过命令 CGI 功能模块向全部主机发出命令的**被授权用户**。列表中的用户同时自动地被授权可以向全部服务发出命令。但列表中的用户并**没有**自动地授权可以查看全部的主机或服务的状态和配置信

息，如果你想让用户同样可以查看状态和配置信息，你需要将用户加入到 authorized_for_all_hosts 变量之中。更多的如何给 CGI 模块设置认证和配置授权的内容可以查阅这个文档。

表 5.134. 全局服务的信息获取权

格式：	**authorized_for_all_services=<user1>,<user2>,<user3>,...<usern>**
举例：	**authorized_for_all_services=nagiosadmin,theboss**

这是一个以逗号分隔的列表，列出了可以查看全部服务的状态和配置的**被授权用户**。但列表中的用户并**没有**自动地授权可以查看全部主机的信息。列表中的用户并**没有**自动地授权向全部服务发送命令。如果你想让这些用户也同样可以发全部服务发送命令，你必须将这些用户加入到 authorized_for_all_service_commands 变量之中。更多的如何给 CGI 模块设置认证和配置授权的内容可以查阅这个文档。

表 5.135. 全局服务的命令操作权

格式：	**authorized_for_all_service_commands=<user1>,<user2>,<user3>,...<usern>**
举例：	**authorized_for_all_service_commands=nagiosadmin**

这是一个以逗号分隔的列表，列出了可以通过命令 CGI 来向全部服务发送命令的**被授权用户**。但列表中的用户并**没有**自动地授权向全部主机发送命令。列表中的用户也**没有**自动地授权查看全部主机的状态和配置信息。如果你想让这些用户同样可以查年全部服务的状态和服务的信息，你必须把这些用户加入到 authorized_for_all_services 变量中。更多的如何给 CGI 模块设置认证和配置授权的内容可以查阅这个文档。

表 5.136. 锁定动作者的用户名

格式：	**lock_author_names=[0/1]**
举例：	**lock_author_names=1**

该选项将使用 WEB 接口时在提交注释、做内容确认和制订宕机计划等操作时限制修改已经他们的动作提交者的名字。如果该选项使能，那么用户在做这些进行命令时将不能修改发出操作者的名字。

1. 0 = 允许用户在提交命令时修改名字
2. 1 = 不许用户提交命令时修改名字(默认值)

表 5.137. 网络拓扑图的背景图设置

格式：	**statusmap_background_image=<image_file>**
举例：	**statusmap_background_image=smbackground.gd2**

该选项将让你可以在使用网络拓扑图时可以指定一个图形文件做为背景图，如果你选择了使用用户定义坐标来绘制的二维网络拓扑图的话。该背景图文件将不能为其他绘制方式提供背景。它假定这个文件是放在图像文件的路径里了(如/usr/local/nagios/share/images)。该路径将自动地在 physical_html_path 域之后加上"/images"生成路径。注意，这个图像文件的格式可以是 GIF、JPEG、PNG 或 GD2 格式。而推荐是 GD2 格式的文件，因为它可以在生成二维图时降低 CPU 负荷。

表 5.138. 默认的二维拓扑图层绘制方式

格式：	**default_statusmap_layout=<layout_number>**
举例：	**default_statusmap_layout=4**

这个选项将让你指定出网络拓扑图 CGI 的默认绘制方式，可用的选项值有：

表 5.139. Statusmap 的<layout_number>取值

Value	Layout Method
0	用户定义坐标系
1	深度图
2	树形折叠图
3	平衡权图
4	圆形图
5	圆形图(出标记的)
6	圆形图(气泡式)

表 5.140. 三维空间的容纳器

格式：	**statuswrl_include=<vrml_file>**
举例：	**statuswrl_include=myworld.wrl**

这个选项将让你指定一个你的对象实体在哪个三维空间的容纳器里展现。它默认是文件已经存放在指定的路径下了，该路径由 physical_html_path 域来指定。注意，这个文件必须是合格的虚拟现实建模(VRML)文件(如你可以在它的专用浏览器里可以查看它)。

表 5.141. 默认三维空间坐标生成算法

格式：	**default_statuswrl_layout=<layout_number>**
举例：	**default_statuswrl_layout=4**

该选项让你指定在三维空间图里对象的三维空间坐标的生成算法。可用的选项值有：

表 5.142. Statuswrl 的<layout_number>取值

值	绘制算法
0	用户定义坐标系
2	折叠树
3	平衡树
4	圆形

表 5.143. CGI 模块的刷新速率

格式：	**refresh_rate=<rate_in_seconds>**
举例：	**refresh_rate=90**

该选项将让你指定以秒为单位的对于 CGI 模块刷新的周期,CGI 模块有状态列表、二维拓扑图和扩展信息等 CGI 模块。

表 5.144. 声音报警

格式：	host_unreachable_sound=<sound_file> host_down_sound=<sound_file> service_critical_sound=<sound_file> service_warning_sound=<sound_file> service_unknown_sound=<sound_file>
举例：	host_unreachable_sound=hostu.wav host_down_sound=hostd.wav service_critical_sound=critical.wav

	service_warning_sound=warning.wav service_unknown_sound=unknown.wav

这个选项将让你指定在查看状态列表时如果有故障发生，你的浏览器里将发出哪个声音文件。如果有故障将按指定的临界故障类型来播放不同的声音文件。这些临界的故障类型是一个或多个主机不可达，至少是一个或多个服务处于未知的状态(见上例中的次序)。声音文件将假定你放在了 HTML 目录的"media/"子目录里(如/usr/local/nagios/share/media)。

表 5.145. Ping 语法

格式：	**ping_syntax=<command>**
举例：	**ping_syntax=/bin/ping -n -U -c 5 $HOSTADDRESS$**

这个选项给出了当从 WAP 接口(使用 statuswml CGI)做 PING 一个主机操作时的 PING 的语法。你必须给出包含全路径名的 PING 的执行文件及全部参数的命令行。命令中使用$HOSTADDRESS$宏来预指定在命令执行前对哪个地址替换并执行 PING 检测。

表 5.146. 扩展 HTML 标记选项

格式：	**escape_html_tags=[0/1]**
举例：	**escape_html_tags=1**

这个选项将决定是否在主机和服务(插件)的检测输出中包含使用 HTML 的扩展选项。如果你使能了它，你的插件将不能使用可点击的超链接标记。

表 5.147. 注释的 URL 指向

格式：	**notes_url_target=[target]**
举例：	**notes_url_target=_blank**

这个选项决定了你的注释 URL 必须要显示的 URL 目标。合法的选项内容包括 _blank、_self、_top、_parent 或是其他合法目标的名字。

表 5.148. 动作的 URL 指向

格式：	**action_url_target=[target]**
举例：	**action_url_target=_blank**

这个选项给定了框内对象的动作里显示的动作 URL 的目标。合法的选项值包括 _blank、_self、_top、_parent 或是任何其他合法目标名字。

表 5.149. Splunk 集成选项

格式：	**enable_splunk_integration=[0/1]**
举例：	**enable_splunk_integration=1**

这 个选项决定了在 WEB 接口里与 Splunk 集成功能是否集成。如果使能它，你页面中将在许多地方呈现出"Splunk It"的链接，CGI 模块页面(日志文件、告警历史、主机和服务的详细信息等)里都有。如果你想对特别的故障发生想知道原诿时很有用。更多关于 Splunk 的信息请访问 http://www.splunk.com/。

表 5.150. Splunk URL

格式：	**splunk_url=<path>**
举例：	**splunk_url=http://127.0.0.1:8000/**

这个选项设置了指向 Splunk 网站的 URL。在 enable_splunk_integration 使能时这个 URL 被 CGI 模块用于指向 Splunk。