第 12 章 开发相关

12.1. 使用内嵌 Perl 解释器

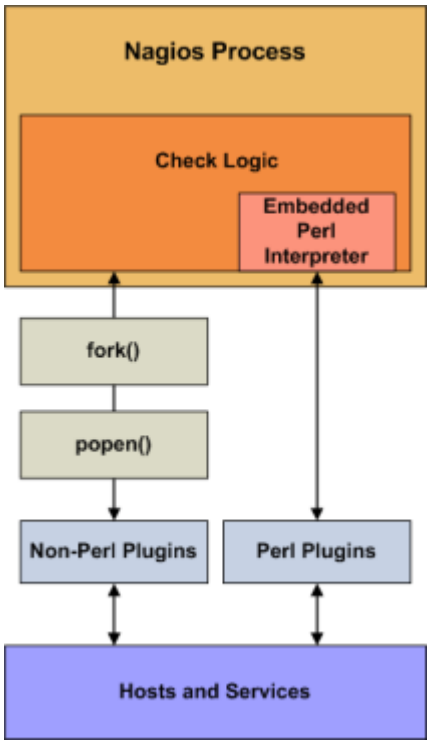
12.1.1. 介绍

Nagios 编译时可以选择支持内嵌 Perl 解释器。这使得 Nagios 可以用更高效率来执行 Perl 所写插件，因而如果严重依赖于 Perl 写的插件的话可能是个好消 息。没有内嵌 Perl 解释器，Nagios 将通过象外部命令一样用派生和执行的方法利用 Perl 所写的插件。当编译中选择了支持内嵌 Perl 解释器 时，Nagios 可以象调用库一样来执行 Perl 插件。

提示

嵌入式 Perl 解释器可以让 Nagios 执行各种 Perl 脚本－不仅仅是插件。本文档只讨论涉及到 Perl 解释里执行对主机与服务检测的插件，但它同样也支持相似的 Perl 脚本以用于其他类型命令(例如通知脚本、事件处理脚本等)。

Stephen Davies 在几年前发布了最初的嵌入式 Perl 解释器，Stanley Hopcroft 是主要的帮助嵌入式 Perl 解释器改进提升的人并且应用它的对优劣性做了批注。他同时给出了有关如何更好地实现用嵌入式 Perl 来实现 Perl 插件的方法。必须注意本文档里用的"ePN"，它指示 Nagios 用 Perl，或是指示了 Nagios 要用嵌入式 Perl 解释器来编译执行它。



12.1.2. 优点

使用 ePN(Nagios 编译有嵌入式 Perl 解释器)的好处有：

1. Nagios 在运行 Perl 插件时将付出更少时间因为它不需要派生进程来执行插件(每次执行要调入 Perl 解释器)。嵌入式 Perl 解释器可以象调用库函数一样来执行插件；
2. 它 会大大降低运行 Perl 插件的系统开销并且(或者同时)可以同时运行更多的 Perl 插件检测。也就是说，可能使用其他语言并没有这些好处，语言象 C/C++、Expect/TCL，用这些语言开发插件公认地比使用 Perl 语言开发插件要慢一个数量级(虽然在最终运行时间上会更快，TCL 语言是个例 外)；
3. 如果不是 C 程序员，仍旧可以用 Perl 来做背负那些繁重的工作而不至于拖慢 Nagios 的运行。但是要注意，ePN 并不能加速插件本身(还要除去内嵌 Perl 解释器的加载时间)。如果要加速插件本身的执行，可以考虑使用 Perl XSUB 包(XS)或是 C，这么做的**前提是**你已经确信插件 Perl 程序足够优化并且保存了合理的算法(用那个 Benchmark.pm 包来比较 Perl 程序模块的性能的方法是**没有意义**的)；
4. 用 ePN 是一个学习 Perl 语言最好的机会。

12.1.3. 缺点

使用 ePN(Nagios 嵌入式 Perl 解释器)比之纯粹 Nagios 程序的缺点相近的，是在运行 Apache 带有 mod_perl(Apache 也是使用嵌入式解释器)和运行纯粹 Apache 程序的两种情形对比，情况也是这样：

1. 一个 Perl 程序可能在使用纯粹 Nagios 程序时**运转良好**但可能在使用 ePN 时却可能**不正常**，要修改 Perl 插件程序以使之运转；
2. 在使用 ePN 时调试 Perl 插件要比使用纯粹 Nagios 调试插件要困难一些；
3. 使用 ePN 的插件比纯粹 Nagios 情况下要更大一些(内存占用)；
4. 想用的一些 Perl 结构体可能不能用或是用起来很困难；
5. 不得不要关注'有多个进程在使用'和选择一种方式看起来更少交换或更少明显交互(注意用解释器执行是并发的－－译者注)；
6. 要有更多的 Perl 功底(但不需要过多 Perl 技巧或素材－除非是使用 XSUBS 的插件)。

12.1.4. 使用嵌入式 Perl 解释器

如果要使用嵌入式 Perl 解释器来运行 Perl 插件和 Perl 脚本，下面这些是需要做的：

1. 带有嵌入式 Perl 解释器支持选项来编译 Nagios(见下面的指令)；
2. 打开主配置文件里的 enable_embedded_perl 选项开关；
3. 设置 use_embedded_perl_implicitly 选项以符合要求。该选项决定默认情况下是否要使用 Perl 解释器来运行个别 Perl 插件和脚本；
4. 偶尔要对某些 Perl 插件或脚本要设置或打开或关闭使用嵌入式 Perl 解释器。这对于部分因带有嵌入式 Perl 解释的 Nagios 在执行 Perl 插件时存在问题时很有用，见下面指令是如何完成的。

12.1.5. 编译一个 Nagios 带嵌入式 Perl 解释器

如果要用嵌入式 Perl 解释器，首先需要编译 Nagios 支持它。只需要运行源程序配置脚本时带上**--enable-embedded-perl** 参数即可。如果要嵌入式解释器缓存编译后的脚本，还要带上**--with-perlcache** 参数，例子：

```
./configure --enable-embedded-perl --with-perlcache
otheroptions...
```

一旦用新选项重新运行配置脚本，一定要重编译 Nagios。

12.1.6. 指定插件使用 Perl 解释器

自 Nagios 的第三版开始，可以在 Perl 插件或脚本中指定是否需要在嵌入式 Perl 解释器里运行。这对于运行那些在 Perl 解释器里运行有问题的 Perl 脚本的处理将非常有帮助。

明确地指定 Nagios 是否要使用嵌入式 Perl 解释器来运行某个 Perl 脚本，把下面的东西加到你的 Perl 插件或脚本里...

如果需要 Nagios 使用嵌入式 Perl 解释器来运行 Perl 脚本，在 Perl 程序里加入下面一行：

```
# nagios: +epn
```

如果*不用*嵌入式解释器，在 Perl 程序里加入下面这一行：

```
# nagios: -epn
```

上面的行必须出现有 Perl 脚本的前 10 行里以让 Nagios 检测 Perl 程序是否要用解释器。

提示

如果**没有明确地**用上述方式指出该插件是否要在 Perl 解释器里运行，Nagios 将自主决定。它取决于 use_embedded_perl_implicitly 变量设置。如果该变量值是 1，全部的 Perl 插件与脚本(那些没明确指定的)将在 Perl 解释器里运行，如果值是 0，那些没明确指定运行方式的插件和脚本将*不会*在解释器里运行。

12.1.7. 开发嵌入式 Perl 解释器可运行的 Perl 插件

更多有关开发嵌入式 Perl 解释器里可运行 Perl 插件的信息可查阅这篇文档。

12.2. 开发使用内嵌式 Perl 解释器的 Nagios 插件

12.2.1. 介绍

Stanley Hopcroft 在嵌入式 Perl 解释器的工作机制方面卓有成效并且给出了利用嵌入式 Perl 解释器方面的优点和缺点的注释，他同时还给出了如何让 Perl 插件更好地在 Perl 解释器里运行的一些提示与帮助，下面内容多来自于他的注解。

需要注意本文档里用的"ePN"，它指定是带有嵌入式Perl解释器的Nagios版本，如果不是，Nagios 需要重新编译以带有嵌入式 Perl 解释器的支持。

12.2.2. 目标受众

1. 中级 Perl 开发者－那些受 Perl 语言强大能力的熏陶却不会利用内在语言特性或是对那些特性掌握不多的开发员；

2. 实用主义者而不是深入研究 Perl 语言特性的；
3. 如果对 Perl 对象、命名空间管理、数据结构和调试等并不很有兴趣，或许也就够了；

12.2.3. 开发 Perl 插件必备知识

不管是否用 ePN 与否，下面的内容要注意：

1. 总是要生成一些输出内容；
2. 加上引用'use utils'并引用些通用模块来输出($TIMEOUT %ERRORS &print_revision &支持等)；
3. 总是知道一些 Perl 插件的标准习惯，如：
    a. 退出时总是 exit 带着$ERRORS{CRITICAL}、$ERRORS{OK}等；
    b. 使用 getopt 函数来处理命令行；
    c. 程序处理超时问题；
    d. 当没有命令参数时要给出可调用 print_usage；
    e. 使用标准的命令行选项开关(象-H 'host'、-V 'version'等)。

12.2.4. 开发 ePN 下的 Perl 插件必做内容

- 1、<DATA>不能使用，用下面方式替换：
- my $data = <<DATA;
- portmapper 100000
- portmap 100000
- sunrpc 100000
- rpcbind 100000
- rstatd 100001
- rstat 100001
- rup 100001
- ..
- DATA
-

```
%prognum = map { my($a, $b) = split; ($a, $b) } split(/\n/, $data) ;
```

- 2、BEGIN 块并不会象你想像的执行，尽可能避免用它；
- 3、确保编译时非常干净，象
    - use strict
    - use perl -w (其他开关特别是[-T]将不能用)
    - use perl -c
- 4、 避免使用全程块里的变量语句(my)来传递到子程序里作为变量数据。如果子程序在插件时被多次调用这将是个致命性错误。这样的子程序块起到一个"终止"作 用，使得全程块里的变量被赋予第一个值而带入到后序子程序调用过程中导致死锁。但

如果全程变量是只读的(比如是个可并发的结构体)就不是问题，在 Bekman 的"推荐替代方式"里是这样来做的：

- 让子程序调用没有副作用方式的参考：

表 12.1. 源程序转换比对

转换前	转换后
my $x = 1 ; sub a { .. Process $x ... } . . a ; $x = 2 ; a ; # anon closures __always__ rebind the current lexical value	 my $x = 1 ; $a_cr = sub { ... Process $x ... } ; . . &$a_cr ; $x = 2 ; &$a_cr ;

- 把全程语句变量送到子程序块里(象个对象或模块那样来处理)；
- 将信息以引用或别名方式送入子程序里(\$lex_var 或$_[n])；
- 用包内全程替换语句并用'use vars qw(global1 global2 ..)'从'use strict'分离出来。

- 5、注意哪里可以得到更多信息。一般信息是要值得怀疑的(O'Reilly 出版的书籍，加上 Damien Conways 著的"Object Oriented Perl")，但真正有用的是从 Stas Bekman 在 http://perl.apache.org/guide/里 所写的 mod_perl 指南，这才是好的起点。那本书非常适用，虽然它里面没有一点关于 Nagios 的东西，全是如果来写在 Apache 里怎么开发利用嵌 入式 Perl 解释器来写 Perl 程序的内容。提倡多使用 perlembed 的在线手册(manpage)。其基础是知道 Lincoln Stein 和 Doug MacEachern 写的有关 Perl 及嵌入 Perl 的一些东西，他们所著的'Writing Apache Modules with Perl and C'一书绝对值得一读。
- 6、注意在 ePN 里插件有可能返回一些奇怪的值，这很有可能是由于前面 4 里所提到的问题所导致的；
- 7、调试前做点准备：
  - 对 ePN 做个检查；
  - 插件里加些打印语句输出到标准错误设备 STDERR(不能用标准输出设备 STDOUT)来显示变量值；

- 在 p1.pl 里加些打印语句来显示 ePN 有要执行插件时是如何看待插件的；
- 在前台模式里运行 ePN(可能要结合前面的一些建议)；
- 插件里用'Deparse'模块来看一下解析命令选项在嵌入解释器实际得到的内容(见 Sean M. Burke 所写的'Constants in Perl'，在 Perl Journal 里，写于 2001 年)；

```
perl -MO::Deparse <your_program>
```

- 8、 还要当心 ePN 把插件转换成什么了，并且如果还有错误可以调试一下转换后的内容。正如下面看到的，p1.pl 把插件当作一个子程序调用，子程序是在一个名 为'Embed::<something_related_to_your_plugin_file_name>'的包里，名字是 'hndlr'。插件可能需要命令参数放在@ARGV 里，因而 pl.pl 会把@_赋予@ARGV。这会按次序从'eval'取出值并且如果 eval 抛出一 个错误(任意一个解析错误或运行错误)都会使插件中断运行。下面打印出了在 ePN 尝试运行之前由它转换好的 **check_rpc** 插件的真正内容，大部分来自于插件的代码没有给出，感兴趣的只是由 ePN 对插件的转换结果。为更清楚起见，转换出的部分用下划线标识了一下：
- ```
  package main;
  use subs 'CORE::GLOBAL::exit';
  sub CORE::GLOBAL::exit { die "ExitTrap: $_[0]
  (Embed::check_5frpc)"; }
  package Embed::check_5frpc; sub hndlr
  { shift(@_);
  @ARGV=@_;
  #! /usr/bin/perl -w
  #
  # check_rpc plugin for Nagios
  #
  # usage:
  #    check_rpc host service
  #
  # Check if an rpc serice is registered and running
  # using rpcinfo - $proto $host $prognum 2>&1 |";
  #
  # Use these hosts.cfg entries as examples
  #
  # command[check_nfs]=/some/path/libexec/check_rpc
  $HOSTADDRESS$ nfs
  #
  service[check_nfs]=NFS;24x7;3;5;5;unix-admin;60;24x7;1;1;1;;che
  ck_rpc
  #
  # initial version: 3 May 2000 by Truongchinh Nguyen and Karl
  DeBisschop
  # current status: $Revision: 1.17 $
  #
  ```
  ```

- # Copyright Notice: GPL
- #
- ...
- rest of plugin code goes here (it was removed for brevity)
- ...

}

- 9、在插件里不用要'use diagnostics'来运行 ePN 方式的程序，这会使得全部插件都返回一个"紧急"状态；
- 10、考虑用一个最小的 C 嵌入 Perl 的程序来检测插件。虽然它还不能担保在它里面运行正常的话也会在 ePN 里运行也能正常，但它可以找到很多 ePN 里肯定也会有的错误。[一个小的 ePN 程序已经在 Nagios 的源程序包里，放在 **contrib/**目录下，可用于测试 ePN 式的 Perl 插件，切换目录到 contrib/下并敲入'make mini_epn'来编译它。它必须与 p1.pl 程序放在同一个目录里运行，p1.pl 也放进了 Nagios 源程序包里了。]

12.3. Nagios 插件 API

12.3.1. 其他资源

如果想给 Nagios 增加一个自己的插件，请访问如下资源：

1. Nagios 插件项目官方网站
2. Nagios 插件开发的官方指南

12.3.2. 插件概览

作为 Nagios 插件的脚本或执行程序必须(至少)要做两件事：

1. 退出时给出几种可能的返回值中的一个；
2. 至少要给出一条输出内容到标准输出设备(STDOUT)。

对 Nagios 来说，插件里面做什么并不重要。自制插件可以是做 TCP 端口状态检测，运行某个数据库查询，检查磁盘空闲空间，或其他需要检测的内容。这取决于你想检测什么东西，这完全由你自己决定。

12.3.3. 返回值

Nagios 用插件的返回值来生成主机或服务的状态。下表里列出了合法的返回值以及对应的服务或主机状态。

表 12.2.

插件返回值	服务状态	主机状态
0	正常(OK)	运行(UP)
1	告警(WARNING)	运行(UP)或宕机(DOWN)/不可达(UNREACHABLE)*
2	紧急(CRITICAL)	宕机(DOWN)/不可达(UNREACHABLE)
3	未知(UNKNOWN)	宕机(DOWN)/不可达(UNREACHABLE)

注意

如果使能 use_aggressive_host_checking 选项，返回值 1 将使主机状态要么是宕机(DOWN)要么是不可达(UNREACHABLE)。其他情况下，返回值 1 将使主机状态是运行(UP)。Nagios 将判定是宕机(DOWN)还是不可达(UNREACHABLE)状态的讨论在这篇文档里有。

12.3.4. 特定插件输出

最小情况下，插件要返回一行文本输出。自 Nagios 3 版本起，插件可以返回多行输出文本。插件可以返回性能数据以让外部应用来做后序处理。输出文本的基本格式如下：

```
TEXT OUTPUT | OPTIONAL PERFDATA

LONG TEXT LINE 1

LONG TEXT LINE 2

...

LONG TEXT LINE N | PERFDATA LINE 2

PERFDATA LINE 3

...

PERFDATA LINE N
```

性能数据(用下划线示意的部分)是可选的，如果插件输出文本里有性能数据，必须用管道符(|)把性能数据与其他数据分开，额外的大段输出行(用文字删除符示意的部分)同样也是可选的。

12.3.5. 插件输出样例

下面看一下插件输出的样例...

案例 1：只有一行文本输出(不带性能数据)

假定插件的输出文本是这样：

DISK OK - free space: / 3326 MB (56%);

如果插件执行的是一个服务检测，整行输出都会保存在$SERVICEOUTPUT$宏里。

案例 2：一行输出带性能数据

插件的输出文本中带有性能数据可给外部应用来处理。性能数据要用管道符(|)分隔开，象是这样：

DISK OK - free space: / 3326 MB (56%); | /=2643MB;5948;5958;0;5968

如果插件执行的是一个服务检测，分隔符左侧的部分将保存在*$SERVICEOUTPUT$*宏里并且右侧(用下划线示意)的部分将保存在*$SERVICEPERFDATA$*宏里面。

案例 3：多行输出(正文和性能数据都有)

插件可以输出多行文本，并且带有正文输出和性能数据，象是这样：

DISK OK - free space: / 3326 MB (56%); | /=2643MB;5948;5958;0;5968

/ 15272 MB (77%);

/boot 68 MB (69%);

/home 69357 MB (27%);

/var/log 819 MB (84%); | /boot=68MB;88;93;0;98

/home=69357MB;253404;253409;0;253414

/var/log=818MB;970;975;0;980

如果插件执行的是一个服务检测，第一行分隔符左侧的部分将保存在$SERVICEOUTPUT$宏里，带有下划线标识的部分(带空格)将保存在$SERVICEPERFDATA$宏里，带删除符标识的部分(不带换行符)的部分将保存在

$LONGSERVICEOUTPUT$宏里(以上的下划线和删除符只是为标记文本段而用的，实际文本中不带有符号格式——译者注)。

每个宏的最终结果是这样的：

表 12.3.

宏	内容
$SERVICEOUTPUT$	DISK OK - free space: / 3326 MB (56%);
$SERVICEPERFDATA$	/=2643MB;5948;5958;0;5968./boot=68MB;88;93;0;98./home=69357MB;253404;253409;0;253414./var/log=818MB;970;975;0;980
$LONGSERVICEOUTPUT$	/ 15272 MB (77%);\n/boot 68 MB (69%);\n/var/log 819 MB (84%);

利用多行输出结果的机制，可以采取多种方式来返回性能数据：

1. 无论什么情况都没有性能数据；
2. 只返回一行性能数据；
3. 只是在后序的行内返回性能数据(第一行不用的管道分隔符右侧不填内容)；
4. 利用全部的输出位置来带出性能数据。

(看起来第一行右侧部分有点"多余"，真的可以不用，但其实这是作者为软件向下兼容低版本使用的插件而特意这么做的，很有必要这么做，看一下源程序就明白了。——译者注)

12.3.6. 插件输出长度限制

Nagios 只处理插件返回的前 4KB 数据内容。这样是为了防止插件返回一个上兆或上千兆的数据块给 Nagios 处理。这个 4K 的限制很容易修改，如果你想改，可以编辑一下源代码里的 MAX_PLUGIN_OUTPUT_LENGTH 宏定义，在源程序包的 include/nagios.h.in 文件里，重编译一下 Nagios 就可以了，其他地方不用动！

12.3.7. 例子

如果想找点例子来学习开发插件，推荐去下载 Nagios 插件项目官方的软件包，插件代码使用多种语言(象 C、Perl 和 SHELL 脚本等)写成插件。取得 Nagios 插件的官方代码包的方法可以看这篇文档。

12.3.8. Perl 插件

Nagios 可以选择支持嵌入式 Perl 解释器可以提高执行 Perl 插件的速度。更多有关使用嵌入式 Perl 解释器来开发 Perl 插件的信息可以查阅这篇文档。