**PHP之PHP-fpm-conf参数详解.**

|  |  |
| --- | --- |
| **[global]** | **说明** |
| pid = run/php-fpm.pid | 默认在安装目录中的var/run/php-fpm.pid |
| error\_log = log/php-fpm.log | 错误日志，默认在安装目录中的var/log/php-fpm.log |
| log\_level = warning | 错误级别. 可用级别为: alert（必须立即处理）, error（错误情况）, warning（警告情况）, notice（一般重要信息）, debug（调试信息）. 默认: notice |
| emergency\_restart\_threshold = 30 | 表示在emergency\_restart\_interval所设值内出现SIGSEGV或者SIGBUS错误的php-cgi进程数如果超过emergency\_restart\_threshold个，php-fpm就会优雅重启。这两个选项一般保持默认值 |
| emergency\_restart\_interval = 60s |
| process\_control\_timeout = 5s | 设置子进程接受主进程复用信号的超时时间. 可用单位: s(秒), m(分), h(小时), 或者 d(天) 默认单位: s(秒). 默认值: 0.  依靠CGI接口，Nginx把收到的请求转给PHP，并从PHP获得返回数据，但CGI实现逻辑是一次请求建立一个PHP进程，处理结束的同时关闭php进程，而php进程的每次启动销毁都很耗资源，于是出现了FastCGI的方式，一个fastcgi进程可以处理多个请求再关闭，但FastCGI依然有缺点，因为fastcgi是单个进程串行，即同一时刻只能处理一个请求，于是php-fpm就出现了，它的作用就是可以管理多个fastcgi。从而实现同时处理多个请求。php-fpm就和nginx的主进程一样，而每一个处理请求的php进程就像是nginx里的多个执行线程。这就是php进程的复用  **php-fpm作为一个调度员，会自动让空闲的FastCGI进程去处理请求，但是在分配请求给php进程前，PHP-FPM需要发送一个进程复用信号给FastCGI，以让FastCGI准备请求处理。但是FastCGI进程并不总是能够处理请求，也就是不能够响应进程复用信号，这个参数就表示了PHP-FPM留给FastCGI进程多久时间去响应进程复用信号，如果超时PHP-FPM会选择其他的fastcgi去处理** |
| daemonize = yes | 后台执行fpm,默认值为yes，如果为了调试可以改为no |
| **[www]** |  |
| listen = /dev/shm/php-cgi.sock | FPM监听端口，即nginx中php处理的地址，可用sock或者127.0.0.1:9000 |
| ;listen = 127.0.0.1:9000 | 第二种设置方法 |
| listen.backlog = -1 | backlog数，-1表示无限制，由操作系统决定，此行注释掉就行 |
| listen.allowed\_clients =127.0.0.1 | 允许访问FastCGI进程的IP，设置any为不限制IP，如果要设置其他主机的nginx也能访问这台FPM进程，listen处要设置成本地可被访问的IP。默认值是any。每个地址是用逗号分隔. 如果没有设置或者为空，则允许任何服务器请求连接 |
| listen.owner = www | socket设置选项，如果使用tcp方式访问，这里注释即可 |
| listen.group = www |
| listen.mode = 0666 |
| user = www | 启动进程的帐户 |
| group = www | 启动进程的帐组 |
| **pm = dynamic（进程管理的三种方式）** | 对于专用服务器，pm可以设置为static 如何控制子进程，选项有static和dynamic和ondemand 如果选择static 则由pm.max\_children指定固定的子进程work数（php-fpm） 如果选择dynamic 则由pm.max\_children、pm.start\_servers、pm.min\_spare\_servers、pm.max\_spare\_servers 参数决定 如果选择dynamic遇到高峰期或者如果pm.process\_idle\_timeout的值太短的话， 无法避免服务器频繁创建进程的问题， 因此pm = dynamic和pm = ondemand谁更适合视实际情况而定 ondemand php-fpm启动的时候，不会给这个pool启动任何一个worker，是按需启动，当有连接过来才会启动 |
| pm.max\_children = 30 | 子进程最大数 一个约30M内存 |
| pm.start\_servers = 20 | 启动时的进程数 |
| pm.min\_spare\_servers = 15 | 保证空闲进程数最小值，如果空闲进程小于此值，则创建新的子进程 |
| pm.max\_spare\_servers = 30 | 保证空闲进程数最大值，如果空闲进程大于此值，此进行清理 |
| **pm.max\_requests = 2048** | 设置每个子进程重启之前服务的请求数.  对于可能存在内存泄漏的第三方模块来说是非常有用的.  如果设置为 ’0′ 则一直接受请求. 等同于 PHP\_FCGI\_MAX\_REQUESTS 环境变量. 默认值: 0. 这段配置的意思是，当一个 PHP-CGI 进程处理的请求数累积到 2048 个后，自动重启该进程。 但是为什么要重启进程呢？ 一般在项目中，我们多多少少都会用到一些 PHP 的第三方库， 这些第三方库经常存在内存泄漏问题，如果不定期重启 PHP-CGI 进程， 势必造成内存使用量不断增长。因此 PHP-FPM 作为 PHP-CGI 的管理器， 提供了这么一项监控功能，对请求达到指定次数的 PHP-CGI 进程进行重启， 保证内存使用量不增长。 正是因为这个机制，在高并发的站点中，经常导致 502 错误， 我猜测原因是 PHP-FPM 对从 NGINX 过来的请求队列没处理好。不过我目前用的还是 PHP 5.3.2，不知道在 PHP 5.3.3 中是否还存在这个问题。 目前我们的解决方法是，把这个值尽量设置大些， 尽可能减少 PHP-CGI 重新 SPAWN 的次数，同时也能提高总体性能。 在我们自己实际的生产环境中发现，内存泄漏并不明显， 因此我们将这个值设置得非常大（204800） 要根据自己的实际情况设置这个值，不能盲目地加大。 |
| **pm.process\_idle\_timeout = 10s** | 空闲worker进程(php-fpm)超时时间,超过10秒就关闭空闲php-fpm |
| **request\_terminate\_timeout = 120（100-60）** | 终止请求超时时间，在worke进程被杀掉之后，提供单个请求的超时间隔。 由于某种原因不停止脚本执行时，应该使用该选项，0表示关闭不启用 当PHP运行在php-fpm模式下，php.ini配置的max\_execute\_time是无效的， 需要在php-fpm.conf中配置另外一个配置项:request\_terminate\_timeout  主要是控制php脚本最大运行时间 |
| request\_slowlog\_timeout = 10s | 当一个请求该设置的超时时间后，就会将对应的PHP调用堆栈信息完整写入到慢日志中. 设置为 '0' 表示 'Off |
| slowlog = var/log/slow.log | 慢请求的记录日志,配合request\_slowlog\_timeout使用 |
| pm.status\_path = /php-fpm\_status | FPM状态页面的网址. 如果没有设置, 则无法访问状态页面. 默认值: none. munin监控会使用到 |
| rlimit\_files = 51200 | 设置文件打开描述符的rlimit限制. 默认值: 系统定义值默认可打开句柄是1024，可使用 ulimit -n查看，ulimit -n 2048修改 |
| rlimit\_core = 0 | 设置核心rlimit最大限制值. 可用值: ‘unlimited' 、0或者正整数. 默认值: 系统定义值 |
| catch\_workers\_output = yes | 重定向运行过程中的stdout和stderr到主要的错误日志文件中. 如果没有设置, stdout 和 stderr 将会根据FastCGI的规则被重定向到 /dev/null . 默认值: 空.配合php.ini可用记录Fastcgi错误日志 |
| ;env[HOSTNAME] = localhost.localdomain | |
| env[PATH] = /usr/local/bin:/usr/bin:/bin | |
| env[TMP] = /tmp |  |
| env[TMPDIR] = /tmp |  |
| env[TEMP] = /tmp |  |