河北师范大学 软件学院
Software College of Hebei Normal University

Linux平台PHP服务端开发——

第六讲 PHP命令行与进程控制

目录

1

PCNTL扩展和POSIX扩展基础

PCNTL扩展

- PCNTL是PHP的进程控制扩展，此扩展只能在Linux/Unix平台上使用，Windows上不可用。PCNTL依赖于Unix方式的进程创建，信号终端等系统调用。
- PHP官方手册：进程控制不能被应用在Web服务器环境，当其被用于Web服务环境时可能会带来意外的结果。
- PHP进行多进程处理的场景可用于服务端的一些运维操作。
- 扩展依赖：PCNTL不依赖其他扩展，PCNTL实际上调用了Linux/Unix提供的系统调用，所以依赖于Linux/Unix平台。
- 安装：编译PHP的时候，configure配置文件加入参数--enable-pcntl，默认是关闭的。

PCNTL基本使用

- PCNTL的简单示例，PHP把系统调用包装成易于使用的PHP扩展。pcntl_fork函数创建一个子进程，成功则在父进程返回子进程的ID，在子进程返回0，失败则返回-1。由于父进程和子进程不同的返回值，可以通过判断返回值让父进程和子进程执行不同的代码。

```php
<?php
$pid = pcntl_fork();
if($pid < 0) {
    exit("Error: pcntl_fork\n");
}
elseif ($pid == 0) {
    echo "Hello, I am child.\n";
}
elseif ($pid>0) {
    echo "I am parent, child $pid running\n";
}
```

等待子进程退出

- 使用pcntl_waitpid挂起当前进程，直到指定的$pid进程退出或是被中断。
- pcntl_waitpid接受三个参数，第一个参数是指定的进程ID：$pid，第二个参数是引用传递一个整型变量保存进程退出状态码。第三个参数$options在提供wait3系统调用的平台上才有效。$options可以是0，或者是：

WNOHANG	如果没有子进程退出立刻返回。
WUNTRACED	子进程已经退出并且其状态未报告时返回。

- 使用方法：pcntl_waitpid($pid, $status);

POSIX扩展

- POSIX扩展是对实现POSIX标准的系统调用接口的封装，此扩展只能在Linux/Unix平台上使用，Windows上不可用。
- 扩展依赖：POSIX扩展不依赖其他扩展。
- 安装：编译PHP的时候，默认开--enable-posix。
- POSIX扩展函数概览：
  - posix_getpid ：获取当前进程的进程ID。
  - posix_getppid ：获取当前进程的父进程的进程ID。
  - posix_getcwd ：返回当前进程的工作目录。
  - posix_kill ：向一个进程发送信号。
  - posix_setsid ：给当前进程分配一个session leader。
  - posix_uname ：获取系统名称。
  - ……
- PCNTL扩展和POSIX扩展往往要配合使用。

POSIX扩展使用示例

- 由于多数函数都比较简单，这里同时编写多个函数的示例。

```php
<?php

echo posix_getpid() . "\n"; //获取进程ID
echo posix_getppid() . "\n"; //获取父进程ID
echo posix_getcwd() . "\n"; //获取当前工作目录
var_dump(posix_uname()); //获取系统名称，版本等信息

```

2

PCNTL扩展信号处理

信号

- 信号是软件中断。很多比较重要的应用程序都需处理信号。信号提供了一种处理异步事件的方法：终端用户键入中断组合键，则会通过信号机制停止一个程序。

- 信号是内核产生的，而生成信号的请求来自三个地方：
  - 用户：通过Ctrl+C、Ctrl+\等产生的信号。
  - 内核：进程出错，内核向进程发送特定信号。
  - 进程：通过系统调用kill向其他进程发送信号。

- 信号都有一个名称，在Linux C编程引入的头文件中，是一个整数值，SIGINT，SIGTERM等名称是一个宏定义。

Linux上的信号与PCNTL的信号

- kill -l会列出所有的信号类型。
- 使用kill –N PID向进程PID发送信号，N是信号的数值。
- 一些信号的意义：

  SIGINT 可以被进程捕获，键盘输入Ctrl+C就会向进程发送SIGINT信号，强制中断程序。
  SIGTERM 可以被进程捕获，kill命令默认的信号，进程捕获后可以做一些清理工作并退出。
  SIGKILL 不可以被捕获，这可以保证进程总是可以被终止，否则进程可以捕获信号并一直运行。
  SIGALRM 定时器时钟信号。
  SIGUSR1/2 进程间通信使用，用户可以自定义。

- PCNTL的信号量定义基本一致，也有一部分新加入的定义，具体可参考PHP官方手册。

注册信号处理函数

- pcntl_signal函数用于信号处理回调函数的注册处理。这个函数接受三个参数，第一个参数是要注册的信号量，第二个参数是信号处理回调函数，第三个参数存在bug，并且设置也无效，实际编码并不使用。

- 目前的PCNTL的信号处理函数使用ticks，要使用declare(ticks=N);才会生效，N是一个整数，表示每执行N条可计时的低级语句，就会检测有无信号事件发生。

- 注册信号处理程序的用法：
  - pcntl_signal(SIGINT, 'signal_handler');
  - pcntl_signal(SIGTERM, function(){……});
  - pcntl_signal(SIGINT, SIG_IGN); //忽略SIGINT信号，SIG_IGN

完整的示例

```php
<?php
declare(ticks=10);
function signal_handler($sig) {
    switch ($sig) {
        case SIGINT:
            echo "get SIGINT signal\n";
            exit(0);
        case SIGTERM:
            echo "get SIGTERM signal\n";
            exit(0);
        default:;
    }
}
pcntl_signal(SIGINT, "signal_handler");
pcntl_signal(SIGTERM, "signal_handler");
echo posix_getpid() . "\n";
while (true) {
    usleep(1000);
}
```

信号处理注意的问题

- 在使用PCNTL处理信号时，没有declare(ticks=N);是不会有效的。
- 调用pcntl_exec执行外部程序，信号会不起作用，原因在于pcntl_exec会调用Linux系统提供的exec*族函数（会根据不同参数执行execv或是execve）。而这些函数会用新的进程执行镜像替换掉当前的，并在执行后退出。
- 调用sleep函数会让程序挂起，信号不会及时被捕获。

3

守护进程

守护进程

- 守护进程是生存期很长的一种进程，通常随系统启动而启动，系统关闭时才终止。
- 守护进程没有控制终端，是在后台运行的。并且父进程是init。
- 通常在shell里面运行一个程序，此程序运行后的父进程是shell，一般这时候shell会等待子进程运行结束，然后回到命令输入模式等待输入新的命令。
- 如果编程实现守护进程，那就必须要脱离shell的控制，并且要把父进程变成init（父进程ID是1）。

使用PCNTL创建守护进程的过程

- Linux的设计是当父进程早于子进程退出，则子进程的父进程变成init。所以在pcntl_fork后，父进程要退出。
- session leader：Linux终端登录后也是有session会话的，一般session leader是shell，因为shell要先运行并等待命令输入，由shell调用fork产生的子进程去运行命令。
- 当pcntl_fork运行后，父进程退出，子进程并不属于某一个shell所控制，父进程已经是init，这时候调用posix_setsid,这个函数创建新的session，并把当前进程作为session leader。
- 在C语言中还要进行工作目录切换，关闭0，1，2文件描述符，重定向0，1，2到/dev/null，但是在PHP中可以不做这些操作。

- 接下来的示例代码并未作错误检测，仅仅是为了表明创建守护进程的过程。并且把工作目录切换到/，在/tmp/pdaemon.pid文件中保存了守护进程的pid。
- 在守护进程创建以后，使用while循环保持进程一直运行，实际开发当中要进行任务处理。

守护进程示例代码

```php
<?php
$pid = pcntl_fork();
if ($pid<0) {
    exit(-1);
}
elseif ($pid > 0) {
    exit(0);
}
file_put_contents('/tmp/pdaemon.pid',posix_getpid());
echo "start daemon ... \n";
posix_setsid();
chdir('/');
while(true) {
    usleep(10000);
}
```

查看守护进程

- 运行ps -o user,pid,ppid,args -e 查看结果，结果显示会找到类似下面的一行。左边显示是wy用户，PID是6415，PPID是1（父进程ID是1），然后是运行的命令。
- 如果结果太多，可以筛选： ps -o user,pid,ppid,args –e | grep php

```
wy          6415     1 php pdaemon.php
```