jinyPHP 3권 - 13. 외부처리

**13. 외부 처리**

====================

PHP는 시스템의 외부 명령어를 실행하고, 결과값을 반환 받을 수 있는 몇개의 시스템 함수들을 제공합니다.

외부 명령 처리는 시스템의 커맨드 명령창에서 실행하는 것과 동일한 처리를 의미합니다.

**13.1 시스템함수**

====================

PHP는 시스템의 명령을 호출하여 실행을 할 수 있습니다. PHP는 3가지 방식으로 시스템을 호출하여 실행을 할 수 있습니다.

**|내부함수|**

|  |
| --- |
| string **exec** ( string $command [, array &$output [, int &$return\_var ]] ) |

내부함수 exec()는 시스템의 외부 **명령을 실행**합니다. exec()는 반환 값으로는 처리 결과 문자열이 반환됩니다.

output 인수가 있으면 지정된 배열은 명령의 모든 출력 행을 배열로 채워집니다. 하지만 \ n 같은 후행 공백은이 배열에 포함되지 않습니다. 만일 output 변수가 초기화 되어 있지 않는 값인경우에 배열 뒤에 추가로 연결이 됩니다.

**예제) exec.php**

|  |
| --- |
| <?php  $result = exec("dir",$output);  echo $result;  echo "<br>";  print\_r($output);  ?> |

**화면출력)**

2개 디렉터리 84,432,617,472 바이트 남음

Array ( [0] => C 드라이브의 볼륨: windows [1] => 볼륨 일련 번호: 447E-0DB0 [2] => [3] => C:\php-7.1.4-Win32-VC14-x86\sample1 디렉터리 [4] => [5] => 2017-08-17 오후 06:06

. [6] => 2017-08-17 오후 06:06

.. [7] => 2017-08-17 오후 06:04 95 exec.php [8] => 1개 파일 95 바이트 [9] => 2개 디렉터리 84,432,617,472 바이트 남음

**|내부함수|**

|  |
| --- |
| string **system** ( string $command [, int &$return\_var ] ) |

내부함수 system()는 외부 프로그램을 실행합니다. 실행후 **결과를 출력**합니다. system()은 주어진 명령을 실행하고 결과를 출력한다는 점에서 함수의 C 버전과 같습니다.

**예제) system.php**

|  |
| --- |
| <?php  $result = system("dir",$output);  echo $result;  ?> |

**화면출력)**

C 드라이브의 볼륨: windows 볼륨 일련 번호: 447E-0DB0 C:\php-7.1.4-Win32-VC14-x86\sample1 디렉터리 2017-08-17 오후 06:20

. 2017-08-17 오후 06:20

.. 2017-08-17 오후 06:04 95 exec.php 2017-08-17 오후 06:20 66 system.php 2개 파일 161 바이트 2개 디렉터리 84,477,603,840 바이트 남음 2개 디렉터리 84,477,603,840 바이트 남음

**|내부함수|**

|  |
| --- |
| string **shell\_exec** ( string $cmd ) |

내부함수 shell\_exec()는 쉘을 통해 명령을 실행합니다. 실행한 전체 출력을 문자열로 반환합니다.

**예제) shell\_exec.php**

|  |
| --- |
| <?php  $output = shell\_exec('ls -lart');  echo "<pre>$output</pre>";  ?> |

**13.2 프로세스**

====================

**|내부함수|**

|  |
| --- |
| resource **proc\_open** ( string $cmd , array $descriptorspec , array &$pipes [, string $cwd [, array $env [, array $other\_options ]]] ) |

내부함수 proc\_open()는 명령 실행과 입 / 출력에 대한 파일 포인터를 생성합니다. proc\_open()은 popen()과 비슷한 점이 많지만 좀더 유연한 제어를 할 수 있습니다.

cmd 는 실행할 명령어 입니다.

두번째 인자 descriptorspec는 인덱스 타입의 배열입니다. 인덱스 키는 자식 프로세스에게 전달되는 방법을 설정합니다. 0은 stdin, 1은 stdout, 2는 stderr입니다.

pipes 는 생성 된 파이프의 PHP 끝 부분에 해당하는 파일 포인터의 인덱스 배열로 설정됩니다.

cwd 는 명령을 실행하는 초기 작업 디렉토리 입니다. 이 값은 상대경로가 아닌 절대 경로 형태로 지정을 해야 합니다. 또는 기본 PHP가 설치된 작업 디렉토리를 사용할 경우에는 NULL을 사용하면 됩니다.

env는 실행되는 명령어에 대한 환경변수를 배열로 설정을 합니다.

other\_options

추가 옵션을 지정할 수 있습니다. 현재 지원되는 옵션은 다음과 같습니다.

suppress\_errors (windows 전용) :이 함수가 TRUE로 설정된 경우 생성 된 오류를 억제합니다.  
bypass\_shell (windows 전용) : TRUE로 설정하면 cmd.exe 셸을 무시합니다.

**|내부함수|**

|  |
| --- |
| int **proc\_close** ( resource $process ) |

내부함수 proc\_open()는 의해 열린 프로세스를 닫습니다. 프로세스의 종료 코드를 반환합니다.

**|내부함수|**

|  |
| --- |
| array **proc\_get\_status** ( resource $process ) |

내부함수 proc\_get\_status()는 proc\_open()에 의해 열린 프로세스에 대한 정보 읽어 옵니다.

* **command** : proc\_open ()에 전달 된 명령 문자열입니다.
* **pid** : process id
* **running** : 프로세스가 아직 실행 중이면 TRUE, 종료 된 경우 FALSE 값을 반환합니다.
* **signaled** : 자식 프로세스가 포착되지 않은 신호에 의해 종료 된 경우 TRUE 값 입니다. Windows에서는 항상 FALSE 값으로 설정됩니다.
* **stopped** : 자식 프로세스가 신호에 의해 중단 된 경우 TRUE 값 입니다. Windows에서는 항상 FALSE로 설정됩니다.
* **exitcode** : 실행도중 프로세스가 반환 한 종료 코드로 FALSE 값 입니다. 첫 번째 호출 만 실제 값을 반환하고, 다음 호출은 -1을 반환합니다.
* **termsig** : 자식 프로세스가 실행을 종료하게 만든 신호의 번호입니다
* **stopsig** : 중지된 경우, 자식 프로세스가 실행을 멈추게 만든 신호의 번호입니다.

**|내부함수|**

|  |
| --- |
| bool **proc\_nice** ( int $increment ) |

내부함수 proc\_nice()는 현재 프로세스의 우선 순위 변경합니다.

proc\_nice()는 increment에 지정된 양만큼 현재 작업중인 프로세스의 우선 순위를 변경하게 됩니다. 양수 값의 증가는 현재 프로세스의 우선 순위를 낮추고, 음수 값의 증가는 우선 순위를 높입니다.

**|내부함수|**

|  |
| --- |
| bool **proc\_terminate** ( resource $process [, int $signal = 15 ] ) |

내부함수 proc\_terminate()는 proc\_open에 의해 오픈된 프로세스를 종료합니다. proc\_open() 함수를 통해서 생성된 프로세스를 종료합니다. proc\_terminate() 호출시 프로세서는 기다리지 않고 즉시 종료합니다.

**예제) proc.php**

|  |
| --- |
| <?php  $descriptorspec = array(  0 => array("pipe", "r"), // stdin  1 => array("pipe", "w"), // stdout  2 => array("file", "/tmp/error-output.txt", "a") // stderr  );  $cwd = '/tmp';  $env = array('some\_option' => 'aeiou');  $process = proc\_open('php', $descriptorspec, $pipes, $cwd, $env);  echo "proc status<br>";  print\_r(proc\_get\_status($process));  echo "<br>";  if (is\_resource($process)) {  // $pipes now looks like this:  // Any error output will be appended to /tmp/error-output.txt  echo "0 => writeable handle connected to child stdin <br>";  $php\_code = "<?php echo 'hello' ?>";  fwrite($pipes[0], $php\_code);  fclose($pipes[0]);  echo "1 => readable handle connected to child stdout <br>";  echo stream\_get\_contents($pipes[1]);  echo "<br>";  fclose($pipes[1]);  // It is important that you close any pipes before calling  // proc\_close in order to avoid a deadlock  $return\_value = proc\_close($process);  echo "command returned $return\_value <br>";  }  ?> |

**화면출력)**

proc status

Array ( [command] => php [pid] => 28647 [running] => 1 [signaled] => [stopped] => [exitcode] => -1 [termsig] => 0 [stopsig] => 0 )

0 => writeable handle connected to child stdin

1 => readable handle connected to child stdout

hello

command returned 0