Search

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

Генераторы »
« Наследование исключений

- Руководство по PHP
- Справочник языка

Change language: Russian

Submit a Pull Request Report a Bug

Fibers

Обзор файберов

(PHP 8 >= 8.1.0)

Файберы представляют собой прерываемые функции полного цикла. Файберы могут быть приостановлены из любого места цикла, приостанавливая выполнение в файбере до тех пор, пока файбер не будет возобновлён в будущем.

Файберы приостанавливают весь цикл выполнения, поэтому вызывающему функцию напрямую не нужно менять способ её вызова.

Выполнение может быть прервано в любом месте цикла с помощью метода Fiber::suspend() (то есть вызов Fiber::suspend() может находиться в глубоко вложенной функции или даже вообще не существовать).

В отличие от генераторов (Generator) без стека, у каждого объекта Fiber есть свой собственный стек вызовов, позволяющий приостанавливать их внутри глубоко вложенных вызовов функций. Функция, объявляющая точку прерывания (то есть вызов метода Fiber::suspend()), не должна изменять свой возвращаемый тип, в отличие от функции, использующей yield, который должен возвращать экземпляр Generator.

Файберы могут быть приостановлены при вызове любой функции, включая те, которые вызываются из виртуальной машины PHP, например, функции, предоставляемые array_map(), или методы, вызываемые foreach для объекта Iterator.

После приостановки выполнение файбера может быть возобновлено с любым значением с помощью метода Fiber::resume() или путём передачи исключения в файбер с помощью Fiber::throw(). Значение возвращается (или выбрасывается исключение) из метода Fiber::suspend().

> **Замечание**: Из-за текущих ограничений невозможно переключать файбер в деструкторе объекта.

Пример #1 Простой пример

```
<?php
$fiber = new Fiber(function (): void {
$value = Fiber::suspend('fiber');
echo "Значение возобновлённого файбера: ", $value, PHP_EOL;
});

$value = $fiber->start();

echo "Значение приостановленного файбера: ", $value, PHP_EOL;

$fiber->resume('test');
?>
```

Результат выполнения приведённого примера:

```
Значение приостановленного файбера: fiber
Значение возобновлённого файбера: test
```


User Contributed Notes 6 notes

62
user at csa dot es ¶
1 year ago
Perhaps not using the same variable name everywhere will be a good idea

```
<?php
$fiber = new Fiber(function (): void {
$parm = Fiber::suspend('fiber');
echo "Value used to resume fiber: ", $parm, PHP_EOL;
});
```

```
$res = $fiber->start();

echo "Value from fiber suspending: ", $res, PHP_EOL;

$fiber->resume('test');
?>
```

up
down
26

Ali Madadi ¶

1 year ago

Here is a simple scheduler and thread pool that implements multithreading using fibers and tick functions in PHP 8.1 and returns the return value of each function in the pool in an array at the end.

Note that due to some bugs, you need to register a new tick function for each "thread". Remember to unregister all of them at the end.

The link bellow is the discussion on a bug that is going on right now (At the time of writing this). Note that based on the discussion, the ability to call Fiber::suspend() inside tick function may become forbidden in PHP 8.2+. But if the bug gets fixed, you can move register_tick_function() line to the top of the class, and this simple multithreading class in pure PHP code will work like a charm.
https://github.com/php/php-src/issues/8960

```
<?php

declare(ticks=1);

class Thread {
protected static $names = [];
protected static $fibers = [];
protected static $params = [];

public static function register(string|int $name, callable $callback, array $params)
{
self::$names[] = $name;
self::$fibers[] = new Fiber($callback);
self::$params[] = $params;
}

public static function run() {
$output = [];

while (self::$fibers) {
foreach (self::$fibers as $i => $fiber) {
try {
if (!$fiber->isStarted()) {
// Register a new tick function for scheduling this fiber
register_tick_function('Thread::scheduler');
$fiber->start(...self::$params[$i]);
} elseif ($fiber->isTerminated()) {
$output[self::$names[$i]] = $fiber->getReturn();
unset(self::$fibers[$i]);
} elseif ($fiber->isSuspended()) {
$fiber->resume();
}
} catch (Throwable $e) {
$output[self::$names[$i]] = $e;
}
}
}
```

```
return $output;
}

public static function scheduler () {
if(Fiber::getCurrent() === null) {
return;
}

// running Fiber::suspend() in this if condition will prevent an infinite loop!
if(count(self::$fibers) > 1)
{
Fiber::suspend();
}
}
}

?>
```

And here is an example code on how to use above Thread class:

```
<?php

// defining a non-blocking thread, so multiple calls will run in concurrent mode using above Thread class.
function thread (string $print, int $loop)
{
$i = $loop;
while ($i--){
echo $print;
}

return "Thread '{$print}' finished after printing '{$print}' for {$loop} times!";
}

// registering 6 Threads (A, B, C, D, E, and F)
foreach(range('A', 'F') as $c) {
Thread::register($c, 'thread', [$c, rand(5, 20)]);
}

// run threads and wait until execution finishes
$outputs = Thread::run();

// print outputs
echo PHP_EOL, '-------------- RETURN VALUES --------------', PHP_EOL;
print_r($outputs);

?>
```

The output will be something like this (but probably different):

```
ABCDEFABCDEFABCDEFABCDEFABCDEFABCEFABFABFABEBEFBEFEFEFAABEABEBEFBEFFAAAAAA
-------------- RETURN VALUES --------------
Array
(
[D] => Thread 'D' finished after printing 'D' for 5 times!
[C] => Thread 'C' finished after printing 'C' for 6 times!
[E] => Thread 'E' finished after printing 'E' for 15 times!
[B] => Thread 'B' finished after printing 'B' for 15 times!
[F] => Thread 'F' finished after printing 'F' for 15 times!
[A] => Thread 'A' finished after printing 'A' for 18 times!
)
```

up
down

16

maxpanchnko at gmail dot com ¶

1 year ago

One of examples, how to make multi_curl faster twice (pseudocode) using Fibers:

```
<?php

$curlHandles = [];
$urls = [
'https://example.com/1',
'https://example.com/2',
...
'https://example.com/1000',
];
$mh = curl_multi_init();
$mh_fiber = curl_multi_init();

$halfOfList = floor(count($urls) / 2);
foreach ($urls as $index => $url) {
$ch = curl_init($url);
$curlHandles[] = $ch;

// half of urls will be run in background in fiber
$index > $halfOfList ? curl_multi_add_handle($mh_fiber, $ch) : curl_multi_add_handle($mh, $ch);
}

$fiber = new Fiber(function (CurlMultiHandle $mh) {
$still_running = null;
do {
curl_multi_exec($mh, $still_running);
Fiber::suspend();
} while ($still_running);
});

// run curl multi exec in background while fiber is in suspend status
$fiber->start($mh_fiber);

$still_running = null;
do {
$status = curl_multi_exec($mh, $still_running);
} while ($still_running);

do {
/**
* at this moment curl in fiber already finished (maybe)
* so we must refresh $still_running variable with one more cycle "do while" in fiber
**/
$status_fiber = $fiber->resume();
} while (!$fiber->isTerminated());

foreach ($curlHandles as $index => $ch) {
$index > $halfOfList ? curl_multi_remove_handle($mh_fiber, $ch) : curl_multi_remove_handle($mh, $ch);
}
curl_multi_close($mh);
curl_multi_close($mh_fiber);
?>
```

up

down

4

nesk at xakep dot ru ¶

1 year ago

I think that in some cases it makes sense to convert a Fiber to a Generator (Coroutine) for convenience. In such cases,

this code will be useful:

```
<?php
function fiber_to_coroutine(\Fiber $fiber): \Generator
{
$index = -1; // Note: Pre-increment is faster than post-increment.
$value = null;

// Allow an already running fiber.
if (!$fiber->isStarted()) {
$value = yield ++$index => $fiber->start();
}

// A Fiber without suspends should return the result immediately.
if (!$fiber->isTerminated()) {
while (true) {
$value = $fiber->resume($value);

// The last call to "resume()" moves the execution of the
// Fiber to the "return" stmt.
//
// So the "yield" is not needed. Skip this step and return
// the result.
if ($fiber->isTerminated()) {
break;
}

$value = yield ++$index => $value;
}
}

return $fiber->getReturn();
}
?>
```

up
down
4

newuser ¶

1 year ago

Example of the same functionality showing what is the difference between Fiber and Generator

```
<?php
$gener = (function () use (&$gener): Generator {
$userfunc = function () use (&$gener) : Generator {
register_shutdown_function(function () use (&$gener) {
$gener->send('test');
});
return yield 'test';
};
$parm = yield from $userfunc();
echo "Value used to resume fiber: ", $parm, PHP_EOL;
})();

$res = $gener->current();
echo "Value from fiber suspending: ", $res, PHP_EOL;
?>
<?php
$fiber = new Fiber(function () use (&$fiber) : void {
$userfunc = function () use (&$fiber) : string {
register_shutdown_function(function () use (&$fiber) {
$fiber->resume('test');
});
return Fiber::suspend('fiber');
```

```
};
$parm = $userfunc();
echo "Value used to resume fiber: ", $parm, PHP_EOL;
});

$res = $fiber->start();
echo "Value from fiber suspending: ", $res, PHP_EOL;
?>
```

up
down
-13
dvohra09 at yahoo dot com ¶
1 year ago

```
Fiber::isRunning does not return a value
Sample
-------

<?php



$fiber = new Fiber(function (): void {
$value = Fiber::suspend('suspend');

echo "Fiber is resumed with value: ", $value, "\n";

});

echo "Fiber not yet started.", "\n";

$value = $fiber->start();

echo "Fiber is started: ", $fiber->isStarted(), "\n";
echo "Fiber is suspended: ", $fiber->isSuspended(), "\n";

echo "Fiber is running: ", $fiber->isRunning(), "\n";

echo "Fiber is suspended with value: ", $value, "\n";
$fiber->resume('resume');

echo "Fiber is running: ", $fiber->isRunning(), "\n";

?>
Output
---

Fiber not yet started. Fiber is started: 1 Fiber is suspended: 1 Fiber is running: Fiber is suspended with value: suspend
Fiber is resumed with value: resume Fiber is running:
```

＋add a note