Speeding up session handling

Native session handling in PHP is fine in most cases. There are at least two possible reasons why you will  
want to change the way sessions are handled:

* When using multiple servers, you need to have common session storage for both servers.
* Default PHP sessions use files, so the maximum performance possible is limited by disk I/O.
* Default PHP sessions are blocking concurrent session storages. In this recipe, we will see how to  
  use efficient storage for Yii sessions.

Подготовка

Создайте новое yii2-app-basic приложение с помощью диспетчера пакетов Composer, как описано в официальном руководстве по адресу  
<http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.   
По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/start-installation>

and install the Memcache server and the memcache PHP extension.

How to do it...

We will stress-test the website using the Apache ab tool. It is distributed with Apache binaries, so if you  
are using Apache, you will find it inside the bin directory.

1. Run the following command replacing your website with the actual hostname you are using:

ab -n 1000 -c 5 http://yii-book.app/index.php?r=site/contact

This will send 1,000 requests, five at a time, and will output stats as follows:

This is ApacheBench, Version 2.3 <$Revision: 1528965 $>

Copyright 1996 Adam Twiss, Zeus Technology Ltd, <http://www.zeustech.net/>

Licensed to The Apache Software Foundation, <http://www.apache.org/>

Server Software: nginx

Server Hostname: yii-book.app

Server Port: 80

Document Path: /index.php?r=site/contact

Document Length: 14866 bytes

Concurrency Level: 5

Time taken for tests: 10.961 seconds

Complete requests: 1000

Failed requests: 0

Total transferred: 15442000 bytes

HTML transferred: 14866000 bytes

Requests per second: 91.24 [#/sec] (mean)

Time per request: 54.803 [ms] (mean)

Time per request: 10.961 [ms] (mean, across all concurrent requests)

Transfer rate: 1375.84 [Kbytes/sec] received

Connection Times (ms)

min mean[+/-sd] median max  
Connect: 0 0 0.0 0 0

Processing: 18 55 324.9 29 4702

Waiting: 15 41 255.1 24 4695

Total: 18 55 324.9 29 4702

We are interested in the requests-per-second metric. The number means that the website can  
process 91.24 requests per second if there are five requests at a time.

Note

Note that debuging is not turned off since we are interested in changes to the session handling  
speed.

1. Now add the following to the /config/web. php components section:

'session' => array(

'class' => 'yii\web\CacheSession',

'cache' => 'sessionCache',

),

'sessionCache' => array(

'class' => 'yii\caching\MemCache',

),

1. Run ab again with the same settings. This time, you should get better results. In my case, it was  
   139.07 requests per second. This means Memcache, as a session handler, performed 52% better than  
   the default file-based session handler.

Note

Don’t rely on the exact results provided here. It all depends on software versions, settings, and  
hardware used. Always try to run all tests yourself in an environment where you are going to  
deploy your application.

1. You can get a significant performance gain by choosing the right session handling backend. Yii  
   supports more caching backends out-of-the-box, including WinCache, XCache, and Zend data  
   cache, which comes with the Zend Server. Moreover, you can implement your own cache backend  
   to use fast noSQL storage, such as Redis.

How it works...

By default, Yii uses native PHP sessions; this means that the filesystem is used in most cases. A  
filesystem cannot deal with high concurrency efficiently.

Memcache or other platforms perform fine in the following situation:

'session' => array(

'class' => 'yii\web\CacheSession',

'cache' => 'sessionCache',

),

'sessionCache' => array(

'class' => 'yii\caching\MemCache',

),

In the preceding config section, we instruct Yii to use CacheSession as a session handler. With this  
component, we can delegate session handling to the cache component specified in cache. This time we  
are using MemCache.

When using a memcached backend, you should take into account the fact that when using these solutions  
the application user can possibly lose the session if the maximum cache capacity is reached.

Note

Note that, when using a cache backend for a session, you cannot rely on a session as a temporary data  
storage, since then there will be no memory to store more data in memcached. In such a case, this will just  
purge all data or delete some of it.

If you are using multiple servers, you cannot use file storage. There is no way to share the session data  
between servers. In the case of memcached, it is easy because it can be easily accessed from as many  
servers as you want.

Also, for sharing the session data you can use DbSession:

return [

// ...

'components' => [

'session' => [

'class' => 'yii\web\DbSession',

],

],

];

Now, create a new table in your database:

CREATE TABLE session (

id CHAR(40) NOT NULL PRIMARY KEY,  
expire INTEGER,  
data BLOB

)

There’s more...

It is a good idea to close the session as soon as possible. If you’re not going to store anything in the  
session during the current request, you can even close it at the very beginning of your controller action.  
This way, even when using files as storage your application should be fine.

Use the following command:

Yii:$app->session->close();

See also

For more information about performance and caching refer to the following URLs:

* [http://www.yiiframework.com/doc-2.0/guide-tutorial-performance-tnning.html](http://www.yiiframework.com/doc-2.0/guide-tutorial-performance-tuning.html)
* <http://www.yiiframework.com/doc-2.0/guide-caching-overview.html>