ApsaraDB for Memcache

Best Practices

Best Practices

Background

You may store some data in the global variable $_SESSION for convenient access when establishing a PHP website. The ini configuration file of PHP contains [Session] configurations for storing the data to a file or the Memcached server. The configuration item session.save_handler = memcached makes the decision. In most cases, the session data does not require persistence. The session information will be cached into Memcached to improve the website performance.

Problem description

ApsaraDB for Memcache and self-built Memcached both comply with the standard Memcached protocol. Some users, in an hope to reduce server memory usage and cut down investment in Memcached maintenance, want to migrate session information stored in the self-built Memcached to ApsaraDB for Memcache without modifying the code. But problems arise during the migration, thus this article which is expected to provide some reference to you.

The most important difference between ApsaraDB for Memcache and self-built Memcached is the “account and password authentication”：

> **ApsaraDB for Memcache**: ApsaraDB for Memcache is a distributed cluster offering external services in a uniform way. It achieves load balancing without SPOF, and you can adjust the configurations flexibly without restarting the service. When external services are involved, the corresponding security mechanism will be in place, such as the white list, flow control, and account and password authentication.
>
> **Self-built Memcached**: Most of the self-built Memcached instances do not require an account or password, so the SASL authentication is missing in self-built Memcached compared with ApsaraDB for Memcache. To migrate session information stored in self-built Memcached to ApsaraDB for Memcache, the account and password need to be configured in php.ini.

Solution

The feature is not supported in older versions of PHP Memcached extensions. You need to upgrade PHP Memcached extension to Version 2.2.0. The example code is as follows:

```
 wget http://pecl.php.net/get/memcached-2.2.0.tgz
tar zxvf memcached-2.2.0.tgz

cd memcached-2.2.0

phpize

./configure --with-libmemcached-dir=/usr/local/libmemcached --enable-memcached-sasl

make

make install
```

Find the just-upgraded memcached.so, run the “stat” command to confirm whether it has been upgraded (pay attention to the modification time).

Modify the php.ini configuration.

Session section: Find the [Session] section and modify the storage engine to:

```
session.save_handler = memcached**(Note the d extension)**
```

Modify the storage address, that is, the Memcache access address, to:

```
session.save_path =
"be6b6b8221cc11e4.m.cnhzalicm10pub001.ocs.aliyuncs.com:11211"(Attention: For the
extension with a “d”, no preceding “tcp://” is needed. For the extension without a “d”,
the preceding “tcp://” is required)
```

Modify the key time for caching to Memcached:

```
session.gc_maxlifetime = 1440(Unit: second. A reasonable life time is strongly recommended to
ensure that only the hotspot data is cached in OCS)
```

Memcached section: In the global section of php.ini, create a separate [memcached] section, and add the configurations below in the blank space:

```
[memcached]
memcached.use_sasl = On
memcached.sess_binary = On
memcached.sess_sasl_username = "your_ocs_name"
```

```
memcached.sess_sasl_password = "your_ocs_password"
memcached.sess_locking = Off
```

The installation steps are completed. For other useful parameters in the above Memcached and Session sections, refer to the links below:

http://php.net/manual/en/memcached.configuration.php

http://php.net/manual/en/session.configuration.php

Next, test whether it works.

Testing

Write the testing code as below in session.php

```
<?php
session_start();
$sn = session_id();
echo "session id:".$sn."\n";
$_SESSION["ocs_key"]="session_value";
echo "session:".$_SESSION["ocs_key"]."\n";
?>
```

The output is as follows:

```
session id:ttrct9coa2q62r2sodlq4qf376

session:session_value
```

Get the data written by session.php from Memcache through the testing code in get.php.

```
<?php
$memc = new Memcached();
$memc->setOption(Memcached::OPT_COMPRESSION, false);
$memc->setOption(Memcached::OPT_BINARY_PROTOCOL, true);
$memc->addServer("be6b6b8221cc11e4.m.cnhzalicm10pub001.ocs.aliyuncs.com", 11211);
$memc->setSaslAuthData("your_ocs_name", "your_ocs_password");
echo $memc->get("memc.sess.key.ttrct9coa2q62r2sodlq4qf376");
/*Attention: Here the key has a prefix defined by the memcached.sess_prefix field in php.ini. The default value is
"memc.sess.key." The prefix is followed by the sessionid "ttrct9coa2q62r2sodlq4qf376" as shown above. */
?>
```

Output:

```
ocs_key|s:13:"session_value";
```

It indicates the PHP SESSION has been successfully written into the Memcache.

Problem description

Some PHP users recently reflected that the performance of ApsaraDB for Memcache fails to meet the expected indexes through tests. We followed up on the situation and found that most users go through the Apache web service to connect to ApsaraDB for Memcache through PHP, using short connections. The overhead of every short connection not only includes socket reconnection, but also complicated re-authentication procedures, being much larger than that of a general request. So it impacts the website efficiency greatly.

Solution

We suggested the users change short connections to persistent connections. But ApsaraDB for Memcache requires the use of PHP Memcached extensions which do not possess the pconnect interface as memcache extensions do. Below is a tutorial on how to establish persistent connections in PHP, for your reference.

On the PHP official website, the introduction of memcached constructor includes a line:

```
Description
Memcached::__construct ([ string $persistent_id ] ): Create a Memcached instance representing the connection to the Memcached service end.
Parameters
Persistent_id: By default, the Memcached instance will be destroyed after the request is over. But you can specify a unique ID for every instance through the persistent_id at the instance creation to share the instance between requests. All the instances created using the same persistent_id share the same connection.
```

That is, you only need to pass an identical persistent_id to the constructor called to achieve shared connections. The implementation in code is as follows:

```
<?php
$memc = new Memcached( ‘ocs’ );//The ocs here is the persistent_id
if (count($memc->getServerList()) == 0) /*Before connection establishment, evaluate first*/
{
echo "New connection"."<br>";

/*All options should be included in the evaluation, because some options may lead to reconnection, or cause a persistent connection to become a short connection.*/
$memc->setOption(Memcached::OPT_COMPRESSION, false);
$memc->setOption(Memcached::OPT_BINARY_PROTOCOL, true);

/* The addServer code must be included in the evaluation, otherwise it is equal to repeating the establishment of the ‘ocs’ connection pool, which may lead to exceptions in the client PHP program.*/
$memc->addServer("your_ip", 11212);
$memc->setSaslAuthData("user", "password");
```

```
}
else
{
echo "Now connections is:".count($memc->getServerList())."<br>";
}
$memc->set("key", "value");
echo "Get from OCS: ".$memc->get("key");
//$memc->quit();/*Do not add ‘quit’ at the end of the code, otherwise persistent connections will be changed to short connections. */
?>
```

Annotations are provided at the three points worth special attention in the above code. The keyword of “ocs” in the constructor is equivalent to a connection pool. Call new Memcached(‘ocs’) and you will be able to obtain the connection from the pool for use.

Execution results

Place the above code under the Apache working path /var/www/html/, and name it test.php. Enter in the browser: Http://your_ip:80/test.php.All the output results of the first eight attempts in the browser are:

```
New connection
Get from OCS: value
```

That is, all these eight attempts create new connections, while the subsequent attempts reuse these eight connections. The packet capture results also show that after the first eight attempts, connections are persistent, with no socket reconnection or authentication.

Why are there eight connections? Check the httpd.conf file and we figure it out: because Apache started eight sub processes:

```
#StartServers：numbers of server processes to start
StartServers 8
```

As a result, eight connections are initialized in the connection pool with the “ocs” persistent_id and later requests can use the eight connections.

Next we test the page navigation. Copy a php file, and establish the persistent_id of the constructor. We still use “ocs”. The result is that the connection is switched (because the called Apache sub processes are different), but with no authentication or socket reconnection. That is, the PHP memcached persistent connection settings are effective. Usually we use the PHP-FPM mode. The FPM process will keep a persistent connection with the memcached server. As a result, the lifecycle of this connection is the same with that of the Apache process.