ApsaraDB for Memcache

Best Practices

Best Practices

Cache PHP session variables

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 is 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 a hope to reduce server memory usage and cut down on investment in Memcached maintenance, want to migrate session information stored in the self-built Memcached to ApsaraDB for Memcache without modifying the code. This document guides you to tackle with the problems, which may arise during the migration process.

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 balance without SPOF, and you can adjust the configuration flexibly without restarting the service. When external services are involved, the corresponding security mechanism is in place, such as the whitelist, throttling, 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 must 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 upgraded memcached.so, run the stat command to confirm whether it has been upgraded (pay attention to the last modified time).

Modify the php.ini configuration.

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

```
session.save_handler = memcached **(Attention: it has a d)**
```

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

```
session.save_path = "be6b6b8221cc11e4.m.cnhzalicm10pub001.ocs.aliyuncs.com:11211"
(Attention: for an extension with a d, no preceding tcp:// is needed. For an 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 make sure that only hotspot data is cached in ApsaraDB for Memcache)
```

Memcached section: In the global section of php.ini, create a separate [memcached] section, and add the following configuration 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 preceding Memcached and Session sections, see the following links.

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

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

Testing

Write the following testing code as 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.

Improve ApsaraDB for Memcache performance through PHP persistent connections

Problem description

Some PHP users experienced 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. This impacts the website efficiency greatly.

Solution

We recommend 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. Following 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 the following:

```
Note:
Memcached::__construct ([ string $persistent_id ] ): Create a Memcached instance representing the connection to the Memcached service end.
Parameter
Persistent_id: By default, the Memcached instance is 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 must pass an identical persistent_id to the constructor called to achieve shared connections. The code implementation is as follows:

```
<?php
$memc = new Memcached( ‘ocs’ );//ocs refers to persistent_id
if (count($memc->getServerList()) == 0) /*Before establishing the connection, first judge*/
{
```

```
echo "New connection"."<br>";

/*All options must 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);
$memc->setOption(Memcached::OPT_TCP_NODELAY, true); //Important: The php memcached has a bug that when the get value does not exist, there is a fixed 40 ms of latency. Enabling this parameter can avoid this bug.

/* 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 is changed to short connections. */
?>
```

Annotations are provided at the three points worth special attention in the preceding code. The keyword of ocs in the constructor is equivalent to a connection pool. Call new Memcached (ocs) and you can obtain the connection from the pool for use.

Execution results

Place the preceding 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, without socket reconnection or authentication.

After checking the httpd.conf file, it is clear that Apache started eight subprocesses and that is the reason for eight connections.

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

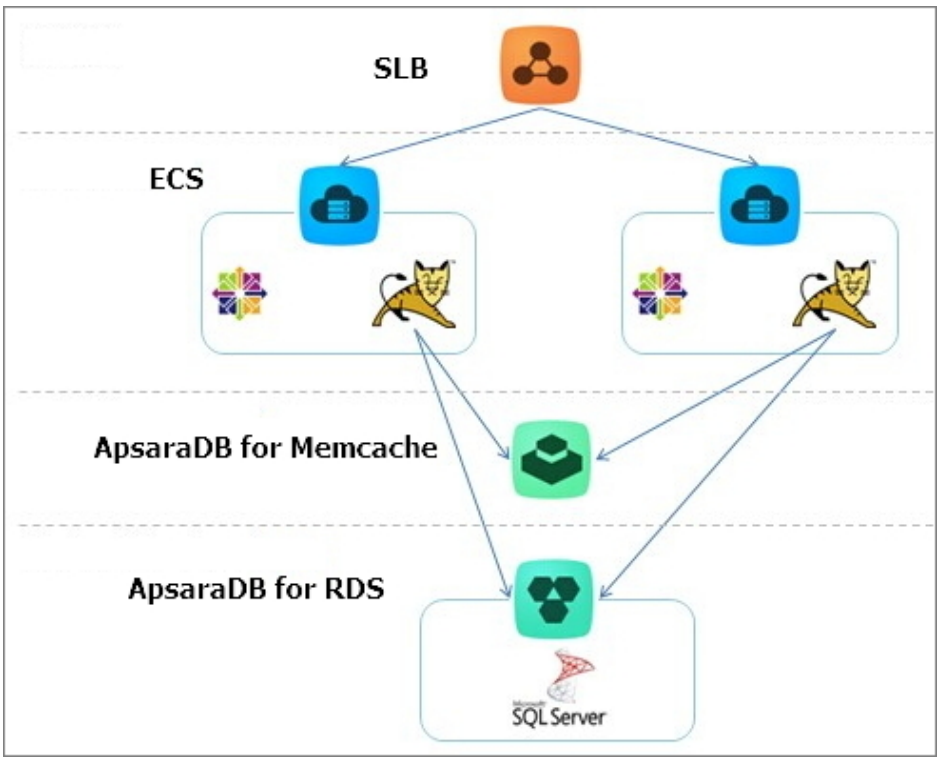
As a result, eight connections are initialized in the connection pool with the ocs persistent_id and later requests can use the eight connections.

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 keeps a persistent connection with the memcached server. As a result, the lifecycle of this connection is the same with that of the Apache process.

Cache Tomcat session variables

Tomcat clusters aim to improve load capacity and evenly distribute accesses to different servers. Take the structure framework illustrated in the following figure as an example.



Alibaba Cloud clusters are generally divided into two modes. One is a fully concentrated session where all nodes are kept consistent with each other, and the other is applicable to a single-session where a specific node in a cluster is designated to provide services.

Install Tomcat and JDK for ECS

Tomcat 7 and JDK 7 are used in this example. We recommend you first download Tomcat to

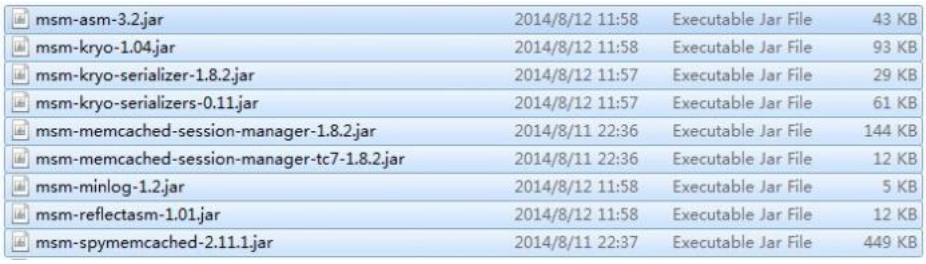
local storage and then upload it to the ECS after configuration is complete.

Tomcat adds Memcached-supported lib package.

For more information on how to configure memcached-session-manager, see this document.

Some lib packages are downloaded in this step. The download addresses are provided in the preceding document. Place the downloaded lib packages in the Tomcat/lib directory, as shown in the following figure.

> **Note**: The prefix of each file msm- does not exist at the beginning. It is added for convenience of management.



With these packages, you can configure a Tomcat connection to Memcache.

Use Tomcat to synchronize sessions to Memcache.

This step requires specific Tomcat configuration. Two configuration modes are available: STICKY and NON-STICKY.

STICKY: The Server Load Balancer distributes all requests to the same cluster node based on user sessions. All session data is obtained from Tomcat, which backs up a session to Memcache to guarantee high efficiency when obtaining a session.

NON-STICKY: The Server Load Balancer distributes requests for every connection separately without considering user sessions. Sessions are all kept in Memcache and all session reads and writes happen in Memcache. This mode requires remote access to data, hence the lower efficiency, but this mode best meets the expected effects of cluster or concentrated cache.

In /Tomcat/conf/context.xml, edit the configuration for connection to Memcache, and add the following configuration descriptions under the <Context> element. Following are the two Tomcat configuration modes:

STICKY mode

```
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="Memcache address: 11211"
username="Memcache instance name"
password="Memcache password"
memcachedProtocol="binary"
sticky="true"
sessionBackupAsync="true"
sessionBackupTimeout="1000"
requestUriIgnorePattern=".*\.(gif|jpg|jpeg|png|bmp|swf|js|css|html|htm|xml|json)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"
></Manager>
```

NON-STICKY mode

```
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="Memcache address: 11211"
username="Memcache instance name"
password="Memcache password"
memcachedProtocol="binary"
sticky="false"
lockingMode="auto"
sessionBackupAsync="false"
sessionBackupTimeout="1000"
requestUriIgnorePattern=".*\.(gif|jpg|jpeg|png|bmp|swf|js|css|html|htm|xml|json)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory" />
```

Note: The memcachedProtocol="binary" setting is required because the attribute value for Tomcat memcached extension mode is text, but Alibaba Cloud only supports binary data.

Modify Tomcat’ s JVM settings and NIO.

In the /Tomcat/bin/ directory, modify JVM settings and add the setenv.sh file. Write the configuration to optimize.

```
CATALINA_OPTS="-server -Xms3072m -Xmx3072m -Xmn1024m -XX:PermSize=96m -
XX:MaxPermSize=128m -XX:+UseConcMarkSweepGC -XX:+UseCMSCompactAtFullCollection -
XX:CMSMaxAbortablePrecleanTime=50 -XX:+CMSPermGenSweepingEnabled"
```

In the /Tomcat/conf/context.xml file, modify NIO settings.

Comment out the original Connector=8080 line and add the following configuration, with NIO mode enabled.

```
<Connector port="8080"
```

```
protocol="org.apache.coyote.http11.Http11NioProtocol"
connectionTimeout="20000"
URIEncoding="UTF-8"
useBodyEncodingForURI="true"
enableLookups="false"
redirectPort="8443" />
```

Create a JSP for checking sessions.

Create a JSP file under the Tomcat/webapps/ROOT directory.

```
 <%@ page language="java" import="java.util.*" pageEncoding="ISO-8859-1"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<base href="<%=basePath%>">
<title>My JSP 'session.jsp' starting page</title>
<meta http-equiv="pragma" content="no-cache">
<meta http-equiv="cache-control" content="no-cache">
<meta http-equiv="expires" content="0">
<meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
<meta http-equiv="description" content="This is my page">
<!--
<link rel="stylesheet" type="text/css" href="styles.css">
-->
</head>
<body>

<h1>
<%
out.println("This is (TOMCAT1), SESSION ID:" + session.getId());
%>
</h1>
</body>
</html>
```

Upload the ready Tomcat to ECS.

After the upload starts, you can see your Tomcat: http://yourserver:8080/session.jsp. If the following text is displayed, it indicates that Tomcat has been successfully connected to Memcache.

```
This is (TOMCAT1), SESSION ID:CAC189E5ABA13FFE29FCB1697F80182B-OCS
```

Note: Memcache can be used normally for caching Tomcat sessions when the website load is low. When traffic load is high and you are encountering frequent session failures, you must upgrade Memcache rules to restore normal functionality.