云数据库 Memcache 版

最佳实践

最佳实践

缓存 PHP session 变量

背景介绍

用户在利用 PHP 搭建网站时，会把一些信息存放在 $_SESSION 全局变量里，可以很方便的存取。在 PHP 的 ini 配置文件里面提供了 [Session] 相关配置，可以支持将信息存到文件或 memcached 服务器里面。由配置项 session.save_handler = memcached 决定。大多数场景，该 session 数据并不需要持久化，且为了提升网站性能，会选择将 session 信息缓存到 memcached 里面。

问题

云数据库 Memcache 版和自建的 memcached 都实现了标准 memcached 协议，用户一方面为了减少服务器内存的占用，一方面减少对 memcached 的维护，希望将 session 的存储从自建的 memcached 迁移到云数据库 Memcache 版上面，且不希望改写代码，切换过程中遇到了问题，因此有了这篇文章，希望能帮到大家。

云数据库 Memcache 版和自建的 memcached 最重要的区别就是“账号密码鉴权”：

> **云数据库 Memcache 版**：分布式集群统一对外提供服务，实现了负载均衡且无单点故障，用户可自由动态弹性调整配置且无需重启服务。既然是对外提供服务，就有相应的安全机制，如白名单、流控、账号密码鉴权。
>
> **自建 memcached**：因为大多数用户自建 memcached 是不需要设置账号密码的，跟云数据库 Memcache 版比就少了 SASL 鉴权流程。那么用户将 session 的存储从自建的 memcached 迁移到云数据库 Memcache 上面，就需要在 php.ini 中配置账号密码。

解决方案

> 在老版本的 php memcached 扩展中无法支持，需要升级 php memcached 扩展至2.2.0版本，示例代码如下：

```
 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
```

找到刚升级 memcached.so，stat 命令确定下是否更新（注意下 modify 时间）。

修改 php.ini 配置。

session段：找到[Session]段落，修改存储引擎为：

```
session.save_handler = memcached**（注意是带d扩展）**
```

修改存储地址，即 Memcache 访问地址为：

```
session.save_path = "be6b6b8221cc11e4.m.cnhzalicm10pub001.ocs.aliyuncs.com:11211"（注意带d扩展，则前面不用加tcp://，不带d的扩展需要加）
```

修改缓存到 memcached 的 key 的时间：

```
session.gc_maxlifetime = 1440（单位是秒，强烈建议必须设置一个合理时间，以保证 OCS 始终只缓存热点数据）
```

memcached 段: 在 php.ini 的全局段，建一个单独段落[memcached]，然后在空白地方加入下面配置:

```
[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
```

安装步骤完结。上述关于 memcached 段和 Session 段其他有用参数参考链接如下：

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

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

接下来是测试是否生效。

测试

写测试代码如下 session.php：

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

输出如下：

```
session id:ttrct9coa2q62r2sodlq4qf376

session:session_value
```

通过测试代码 get.php 从 Memcache 获取刚才 session.php 写入的 session 数据。

```
<?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");
/*注意这里的key是有前缀的，由php.ini中memcached.sess_prefix字段决定，默认值为"memc.sess.key."。然后再拼接上面打出来的sessionid“ttrct9coa2q62r2sodlq4qf376”即可。*/
?>
```

该代码输出如下：

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

即 PHP SESSION 已经成功写入 Memcache。

利用PHP长连接提高性能

问题介绍

最近有 PHP 用户反馈对云数据库 Memcache 版做性能测试的结果，达不到预期的性能指标。通过了解具体情况，大多数用户在使用 PHP 连接云数据库 Memcache 版时，都是通过走 Apache WEB 服务再连云数据库 Memcache 版，使用的是短连接。而每个短连接的开销不止是 socket 重连，还有复杂的重新鉴权流程，开销比一个普通请求大许多，因此对网站的效率是有很大影响的。

解决方案

于是我们建议用户改短连接为长连接，但是云数据库Memcache要求使用的PHP MEMCACHED扩展，不像memcache扩展那样有个pconnect接口。如何才能在PHP中建立长连接，以下教程供大家参考。

在PHP 官网介绍 memcached 构造函数时有下面一段话：

```
说明
Memcached::__construct ([ string $persistent_id ] )创建一个代表到Memcached服务端连接的Memcached实例。

参数
persistent_id默认情况下，Memcached实例在请求结束后会被销毁。但可以在创建时通过persistent_id为每个实例指定唯一的ID， 在请求间共享实例。所有通过相同的persistent_id值创建的实例共享同一个连接。
```

即在调用构造函数时传给它一个同样的 persistent_id 就能实现共享连接。代码实现如下：

```
<?php
$memc = new Memcached（‘ocs’);//这里的ocs，就是persistent_id
if (count($memc->getServerList()) == 0) /*建立连接前，先判断*/
{
echo "New connection"."<br>";

/*所有option都要放在判断里面，因为有的option会导致重连，让长连接变短连接！*/
$memc->setOption(Memcached::OPT_COMPRESSION, false);
$memc->setOption(Memcached::OPT_BINARY_PROTOCOL, true);
$memc->setOption(Memcached::OPT_TCP_NODELAY, true); //重要，php memcached有个bug，当get的值不存在，有固定40ms延迟，开启这个参数，可以避免这个bug

/* addServer 代码必须在判断里面，否则相当于重复建立’ocs’这个连接池，可能会导致客户端php程序异常*/
$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();/*代码结束的地方一定不能加quit，否则变短连接！*/
?>
```

上述代码要特别注意的三个地方都加了注释。构造函数里的“ocs”关键字，就相当于一个连接池了，需要使用的连接调用 new Memcached(‘ocs’)就能从池里获取连接。

执行结果

将上述代码放到 Apache 工作路径/var/www/html/下面，命名为 test.php。然后再浏览器输入：http://your_ip:80/test.php 。前8次浏览器都输出结果为：

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

即这8次都是新建连接，而8次后都是复用这8个链接了。抓包结果也显示8次后都是长连接，无 socket 重连无鉴权。

那么为什么有8个链接? 通过查看 httpd.conf 配置文件，是因为 appache 启动了8个子进程：

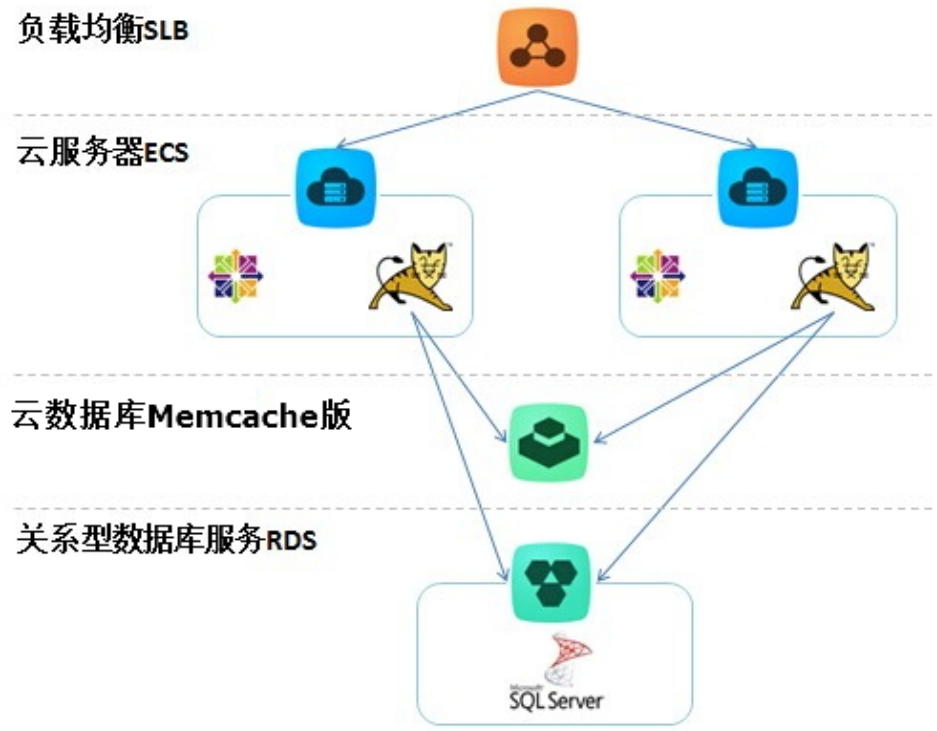
```
#StartServers：numbers of server processes to start
StartServers 8
```

于是刚好在‘ocs’这个 persistent_id 的连接池里面初始化8个连接，此后的请求就用这8个链接了。

接下来测试页面跳转，拷贝一个 php 文件，建立连接的构造函数的 persistent_id 还是用 “ocs”。得到的结果是从一个连接换到了另一个连接上（因为调用的 Apache 子进程不一样），但无鉴权无 socket 重连过程。即 PHP memcached 的长连接设置是有效的。通常我们使用的都是 PHP-FPM 模式， FPM 进程会和 memcached server 保持长连接，因此该连接的生命周期同 Apache 进程。

缓存 Tomcat session 变量

做 Tomcat 集群的目的是为了提供更高的负载能力，把访问均摊到不同的服务器上。以下图的结构框架作为例。



阿里云集群一般分为两种模式，一种是完全集中 SESSION，各个集群点保持一致；还有一种就是基于一次会话指定某一个集群中节点提供服务。

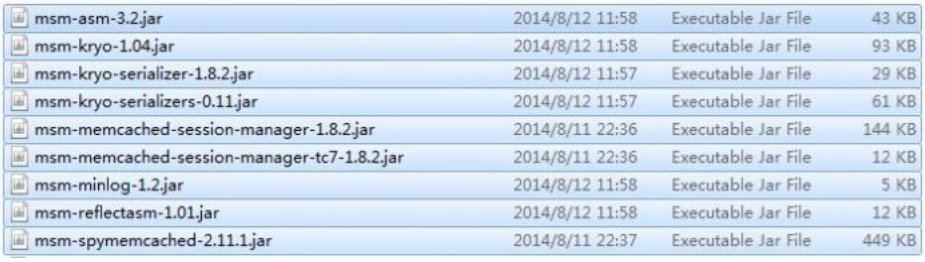
ECS 安装 TOMCAT 和 JDK。

本例中用的是TOMCAT 7 以及 JDK 7。建议 TOMCAT 先下载到本地，配置好了再上传到 ECS。

TOMCAT 增加 memcached 支持的 lib 包。

配置 memcached-session-manager，请参考该文档。

本步骤主要下载一些 lib，下载地址在上面的文档中有给出。把下载到的 lib 放到 Tomcat/lib 目录里面， 如下图所示。（**注意**：每个文件的前缀“msm-”本来是没有的，是为了方便管理加的。）



msm-asm-3.2.jar	2014/8/12 11:58	Executable Jar File	43 KB
msm-kryo-1.04.jar	2014/8/12 11:58	Executable Jar File	93 KB
msm-kryo-serializer-1.8.2.jar	2014/8/12 11:57	Executable Jar File	29 KB
msm-kryo-serializers-0.11.jar	2014/8/12 11:57	Executable Jar File	61 KB
msm-memcached-session-manager-1.8.2.jar	2014/8/11 22:36	Executable Jar File	144 KB
msm-memcached-session-manager-tc7-1.8.2.jar	2014/8/11 22:36	Executable Jar File	12 KB
msm-minlog-1.2.jar	2014/8/12 11:58	Executable Jar File	5 KB
msm-reflectasm-1.01.jar	2014/8/12 11:58	Executable Jar File	12 KB
msm-spymemcached-2.11.1.jar	2014/8/11 22:37	Executable Jar File	449 KB

有了这些包之后就可以配置 TOMCAT 连接到 Memcache。

TOMCAT 同步 Session 到 Memcache。

该步骤要做 TOMCAT 具体配置了，有两种配置模式：STICKY 和 NON-STICKY。

STICKY：负载均衡会根据用户会话，每次都分配到同一个集群节点上。用户每次获取会话数据都是从 TOMCAT 里面取得，TOMCAT 会备份一个 SESSION 到 Memcache，这样可以保持最高效地获取 SESSION。

NON-STICKY：负载均衡不会管用户会话，而是按每次连接分别分发的方式，会话都保持在 Memcache 上，每次读写都在 Memcache 上。由于会远程访问数据，所以效率会低一些，但是这种却是最符合集群或集中缓存预期效果的。

在 /Tomcat/conf/context.xml 中编辑连接到 Memcache 的配置，在 <Context> 元素下增加下述的配置描述即可。下面分别是 TOMCAT 的两种配置模式：

STICKY 模式

```
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="Memcache 的地址:11211"
username="Memcache 实例名"
password="Memcache 密码"
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 模式

```
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="Memcache 的地址:11211"
username="Memcache 实例名"
password="Memcache 密码"
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" />
```

注意：必须要有memcachedProtocol="binary"，因为 Tomcat memcached 插件模式这个属性值是 text，而阿里云支持的是 binary 模式的数据。

修改 TOMCAT 的 JVM 设置以及 NIO。

在 /Tomcat/bin/ 目录下修改 JVM 设置，增加 setenv.sh 文件，写定要优化的配置。

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

在 /Tomcat/conf/context.xml 文件中修改 NIO 设置。

注释掉原有的 Connector=8080 的定义，增加如下的配置定义，使用 NIO 方式。

```
 <Connector port="8080"
protocol="org.apache.coyote.http11.Http11NioProtocol"
connectionTimeout="20000"
URIEncoding="UTF-8"
useBodyEncodingForURI="true"
enableLookups="false"
redirectPort="8443" />
```

创建一个检查 SESSION 的 JSP。

创建一个 JSP 文件到 Tomcat/webapps/ROOT 目录下。

```
 <%@ 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>
```

上传配置好的 TOMCAT 到 ECS。

上传后启动，就可以看见你的 TOMCAT 了：<服务器地址>:8080/session.jsp。 如看见的是如下字样，说明 TOMCAT 连接 Memcache 成功了。

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

注意： 在网站负载较低情况下，能正常使用 Memcache 来缓存 tomcat session。如果负载较高，即发现 session 频繁失效，需要升级 Memcache 规格才能正常使用。