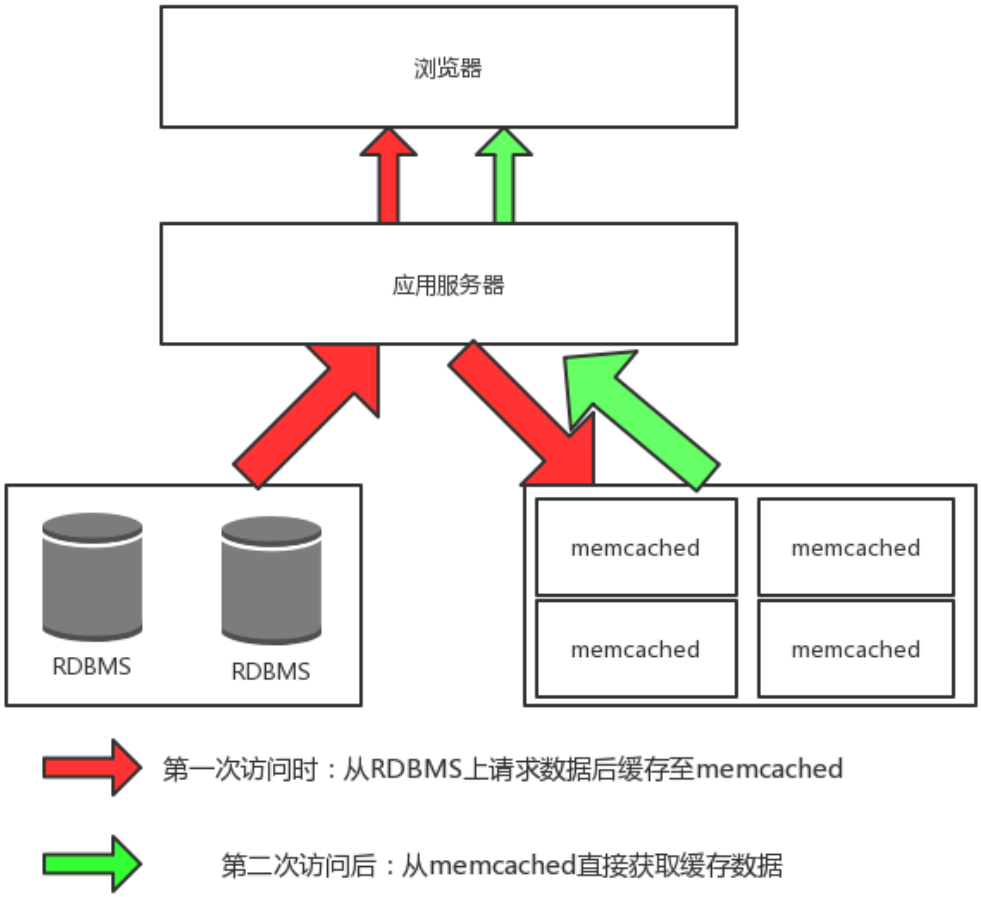
MemCache 介绍

MemCache 是一套高性能分布式的高速缓存系统，用于动态 Web 应用以减轻数据库负载，由 LiveJournal 的 Brad Fitzpatrick 开发。目前被许多网站使用以提升网站的访问速度，尤其对于一些大型的、需要频繁访问数据库的网站访问速度提升效果十分显著。这是一套开放源代码软件，以 BSD license 授权发布。

MemCache 通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高了网站访问的速度。 MemCache 是一个存储键值对的 HashMap，在内存中对任意的数据（比如字符串、对象等）所使用的 key-value 存储，数据可以来自数据库调用、API 调用，或者页面渲染的结果。

MemCache 设计理念就是小而强大，它简单的设计促进了快速部署、易于开发并解决面对大规模的数据缓存的许多难题，而所开放的 API 使得 MemCache 能用于 Java、C/C++/C#、Perl、Python、PHP、Ruby 等大部分流行的程序语言。



memcache 与 memcached 区别

1. MemCache：

MemCache 这个软件项目一般叫 MemCache，但项目的主程序文件叫 memcached.exe。MemCache 的工作机制是在内存中开辟一块空间，然后建立 HashTable，依靠服务端的守护进程管理这些 HashTable。由于这个命名问题，这个软件系统被所以很多人叫做 MemCache，也可以被称为 MemCached。

2. MemCached：

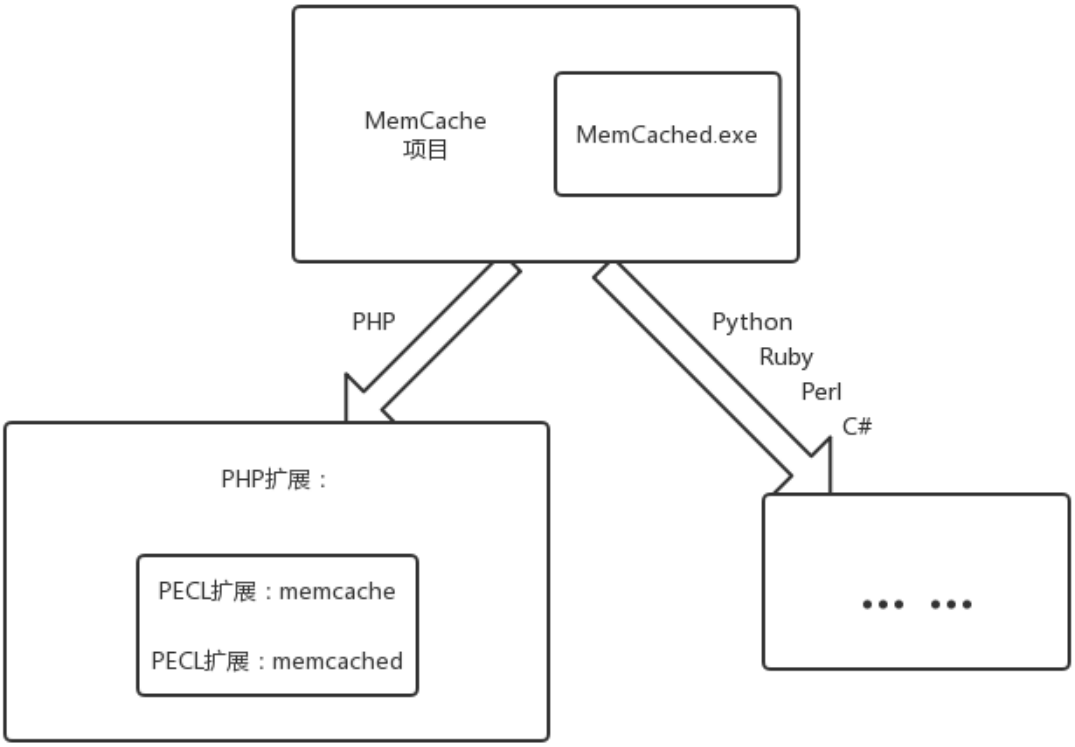
MemCache 是该系统的项目名称，MemCached 是该系统的主程序文件（字母 d 可以理解为 daemon），以守护程序方式运行于一个或多个服务器中，随时接受客户端的连接操作，使用共享内存存取数据。

3. memcache 客户端（PHP）：

PHP 有两个 memcache 客户端：php memcache 和 php memcached。

对于这个内存缓存系统，PHP 有两个扩展，分别是 memcache 和 memcached 扩展。而 memcached 和 memcache 的守护进程 memcached 同名，比较容易引起混淆。

php memcache 是完全在 PHP 框架内开发的，memcache 是原生实现的。OO 和非 OO 两套接口并存。php memcached 是使用 libmemcached 的。只支持 OO 接口，随着 memcached 服务器端的改进，这个 lib 也必定会马上跟进。memcached 支持 Binary Protocol，而 memcache 不支持，意味着 memcached 会有更高的性能。不过，还需要注意的是，memcached 目前还不支持长连接。



MemCache 工作原理

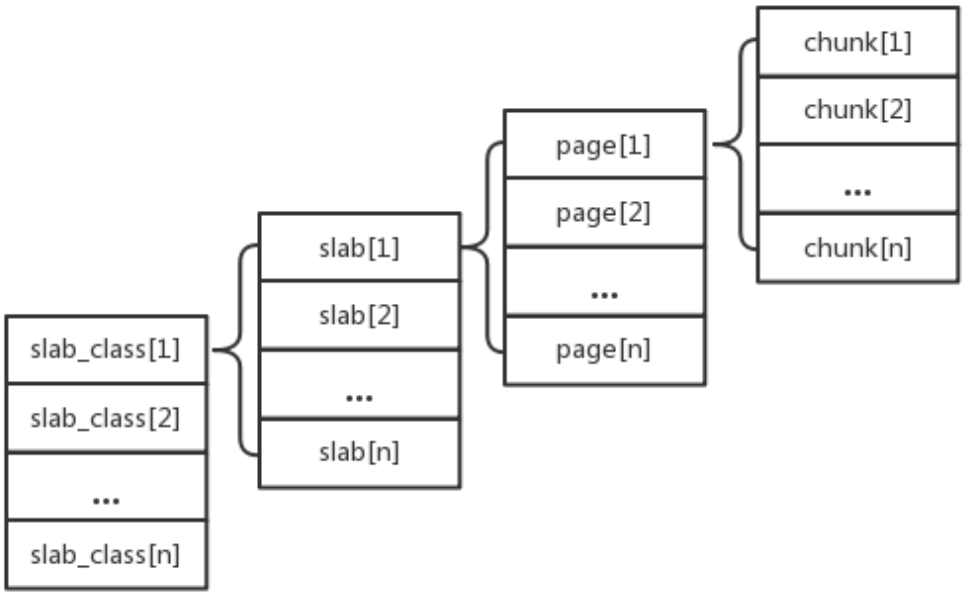
MemCache 采用 C/S 架构，在服务器端启动后，以守护程序的方式，监听客户端的请求。启

动时可以指定监听的 IP（服务器的内网 ip/外网 ip）、端口号（所以做分布式测试时，一台服务器上可以启动多个不同端口号的 MemCached 进程）、使用的内存大小等关键参数。一旦启动，服务就会一直处于可用状态。

为了提高性能，MemCache 缓存的数据全部存储在 MemCache 管理的内存中，所以重启服务器之后缓存数据会清空，不支持持久化。

1. MemCache 内存管理

• 内存结构



1) 每个 slab_class 里面包含若干个 slab。

2) 每个 slab 里面包含若干个 page，page 的默认大小是 1M。

3) 每个 page 里面包含若干个 chunk，chunk 是数据的实际存放单位，每个 slab 里面的 chunk 大小相同

• 内存分配方式

1) Memcached 使用 slab allocation 机制来分配和管理内存。

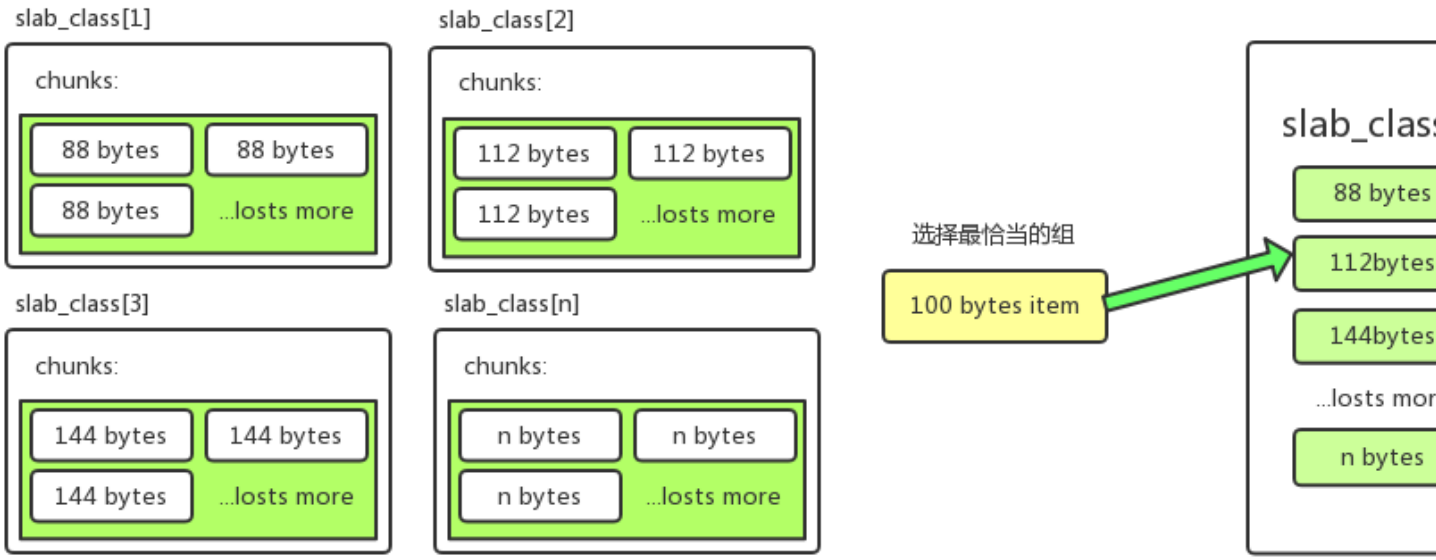
先将分配的内存按照预设好的大小分割成特定长度的内存块，再把尺寸相同的内存块分

成组。数据在存放时，根据键值大小去匹配 slab 大小，找就近的 slab 存放。而传统的内存管理方式是，使用完通过 malloc 分配的内存后通过 free 来回收内存，这种方式容易产生内存碎片并降低操作系统对内存的管理效率。

2) 存放数据时，首先 slab 要申请内存，申请内存以 page 为单位。

在放入第一个数据的时候，无论大小为多少，都会有 1M 大小的 page 被分配给该 slab。申请到 page 后，slab 会将这个 page 的内存按 chunk 的大小进行切分，这样就变成了一个 chunk 数组，最后从这个 chunk 数组中选择一个用于存储数据。

示例：



本例中，slab[1]的 chunk 大小为 88 字节、slab[2]的 chunk 大小为 112 字节、slab[3]的 chunk 大小为 144 字节...（默认相邻 slab 内的 chunk 基本以 1.25 为比例进行增长，MemCache 启动时可以用-f 指定这个比例），那么一个 100 字节大小的 value，将被放到 slab[2]中。

这是由于在 MemCache 中，value 的存放位置由其大小决定，value 会被存放到与 chunk 大小最接近的一个 slab 中，即 slab allocation 机制。

• 内存回收方式

由于 slab allocator 机制中，分配的 chunk 大小是固定的，因此可能会造成内存的浪费。对

于 chunk 空间的浪费问题,无法彻底解决,只能缓解该问题。

1) 不同于 FIFO(First In,First Out)删除机制，当数据容量用完 MemCached 分配的内存后，就会基于 LRU(Least Recently Used)算法清理失效的缓存数据（放入数据时可设置失效时间），或者清理最近最少使用的缓存数据，然后放入新的数据。

2) 在 LRU 中，MemCached 使用的是一种 Lazy Expiration 策略，不会监控存入的 key/value 对是否过期，而是在获取 key 值时查看记录的时间戳，检查 key/value 对空间是否过期，这样可减轻服务器的负载，即过期数据惰性删除，节省了 cpu 时间和检测成本。

3) 如果 MemCache 启动没有追加-M，则表示禁止 LRU，这种情况下内存不够时会报 Out Of Memory 错误。

2. MemCache 分布式存储

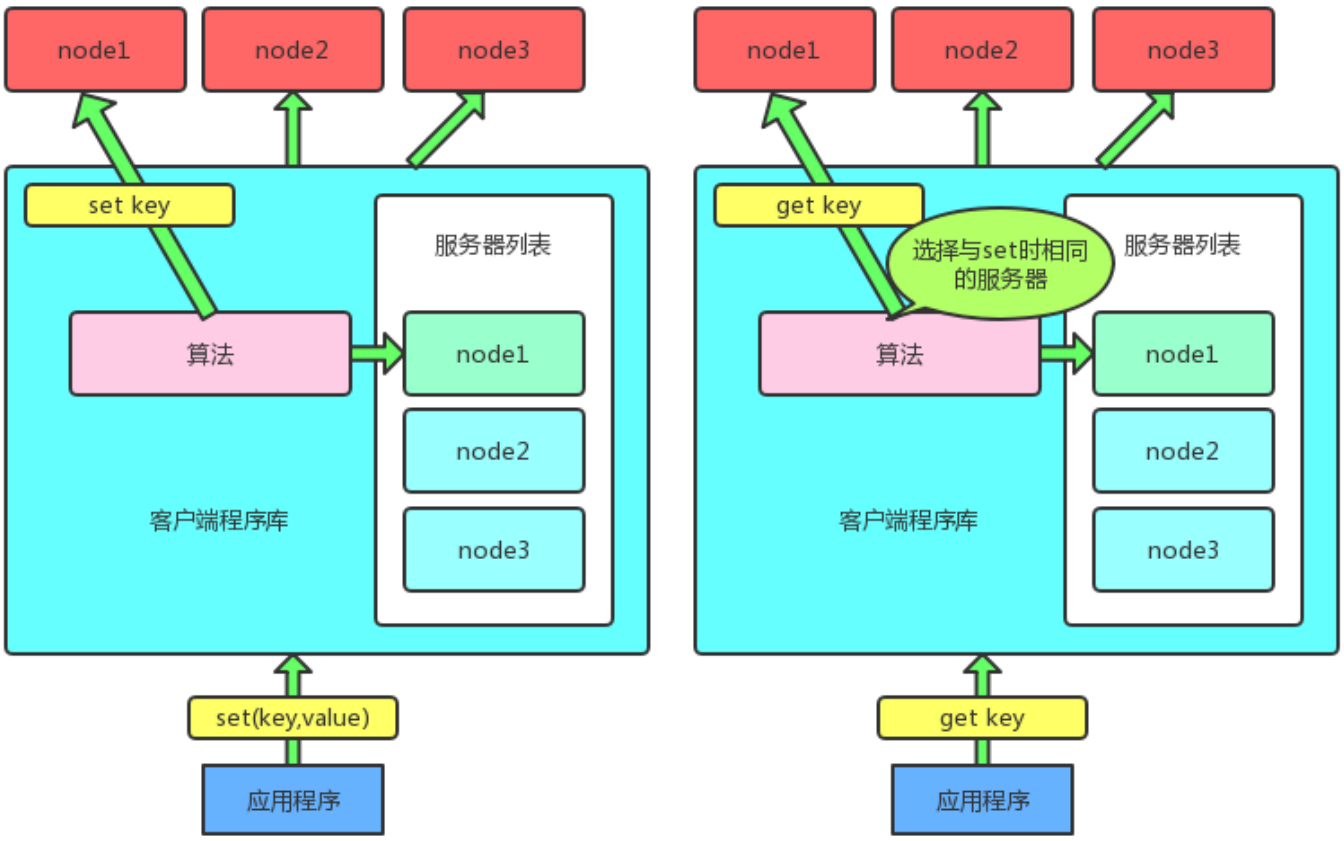
为了提升 MemCache 的存储容量和性能，可能会有多个 MemCache 服务器来对客户端提供服务，这就是 MemCache 的分布式。

• 分布式实现原理

MemCache 采用了单进程、单线程、异步 I/O，基于事件(event_based)的服务方式，使用 libevent 作为事件通知实现分布式存储。

MemCache 当中的 Server 可以协同工作，但他们之间保存的数据各不相同，而且并不通信，每个 Server 只管理自己的数据。服务端并没有“分布式”功能。 Client 端通过 IP 地址和端口号指定 Server 端，将需要缓存的数据以 key-value 的对应形式保存在 Server 端。key 值通过 hash 进行转换，根据 hash 值把 value 传递到对应的具体的某个 Server 上。

当需要获取对象数据时，也根据 key 进行获取。首先对 key 进行 hash 转换，通过获得的值可以确定它被保存在了哪台 Server 上，然后再向该 Server 发出请求。Client 端只需要知道保存 hash(key)的值在哪台服务器上即可。



向 MemCache 集群存入/取出 key/value 时，MemCache 客户端程序根据一定的算法计算存入哪台服务器，然后再把 key/value 值存到此服务器中。

亦即是说，存取数据分为二步：1. 选择服务器；2. 存/取数据。

• 分布式算法解析

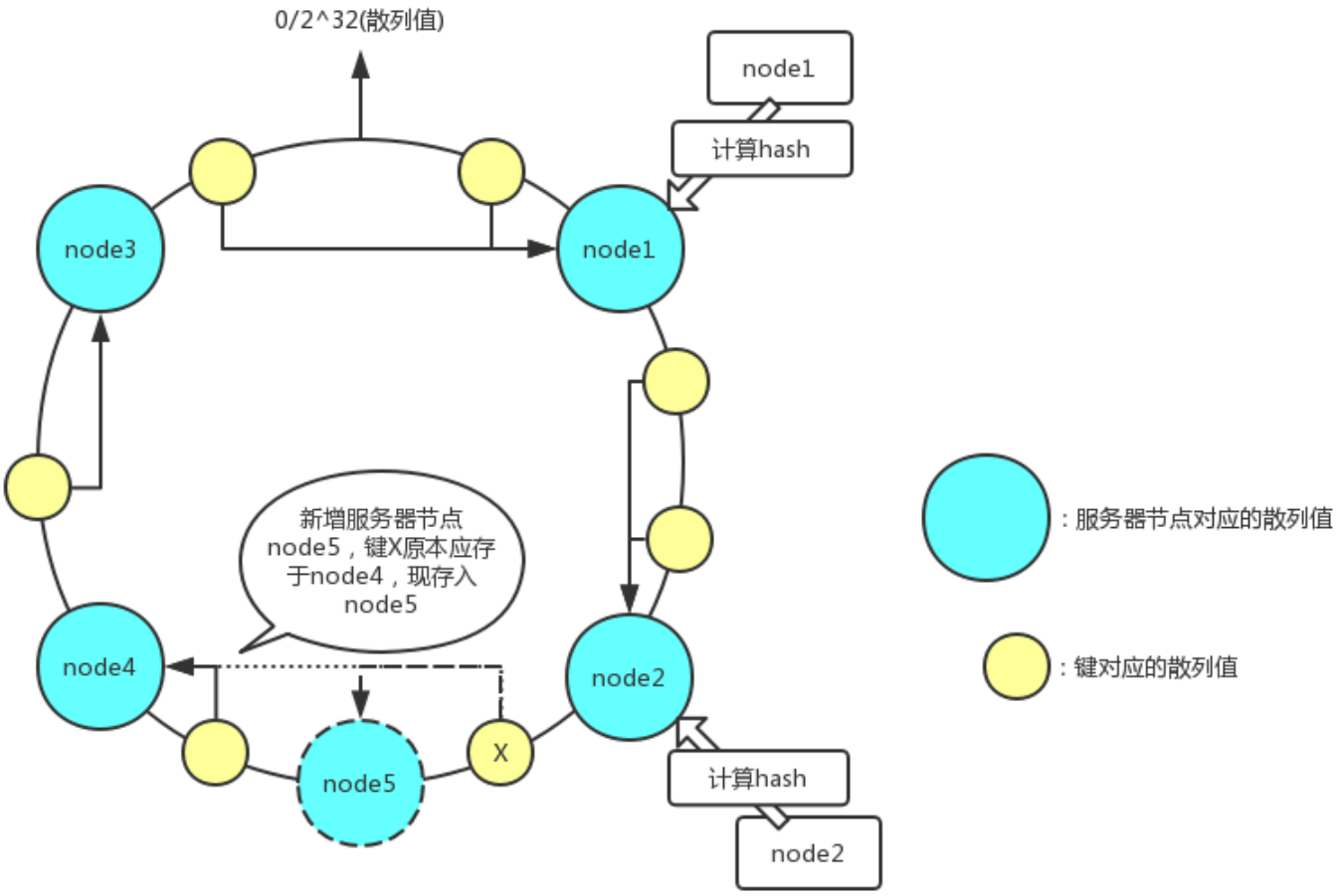
1) 余数算法：

根据键的整数散列值（键 string 的 HashCODE 值），除以服务器台数，根据余数确定存取服务器，这种方法计算简单，高效，但在 MemCache 服务器数量变化时，几乎所有的缓存都会失效。

2) 散列算法：

先算出 MemCache 服务器的散列值，并将其分布到 0 到 2^32 的圆上，然后用同样的方法算出存储数据的键的散列值并映射至圆上，最后从数据映射到的位置开始顺时针查找，将数据保存到查找到的第一个服务器上，如果超过 2^32，依然找不到服务器，就将数据保存到第

一台 MemCache 服务器上。如果添加了一台 MemCache 服务器，只在圆上增加服务器的逆时针方向的第一台服务器上的键会受到影响。



3. MemCache 线程管理

MemCache 网络模型是典型的单进程多线程模型。采用 libevent 处理网络请求，主进程负责将新来的连接分配给 work 线程，work 线程负责处理连接，有点类似于负载均衡，通过主进程分发到对应的工作线程。

MemCache 默认有 7 个线程，4 个主要的工作线程，3 个辅助线程，线程可划分为以下 4 种：

1) 主线程：负责 MemCache 服务器初始化，监听 TCP、Unix Domain 连接请求；

2) 工作线程池：MemCache 默认 4 个工作线程，可通过启动参数修改，负责处理 TCP、UDP，Unix 域套接口链路上的请求；

3) assoc 维护线程：MemCache 内存中维护一张巨大的 hash 表，该线程负责 hash 表动态

增长；

4) slab 维护线程：即内存管理模块维护线程，负责 class 中 slab 的平衡，MemCache 启动选项中可关闭该线程。

MemCache 特性与限制

1. MemCache 中可以保存的 item 数据量是没有限制的，只要内存足够。

2. MemCache 单进程在 32 位机中最大使用内存为 2G，在 64 位机中则没有限制。

3. Key 的最大长度为 250 个字节，超过该长度将无法存储，由常量 KEY_MAX_LENGTH 250 控制。

4. 单个 item 最大数据是 1MB，超过 1MB 的数据不予存储，由常量 POWER_BLOCK 1048576 进行控制。

5. MemCache 服务端是不安全的。比如已知某个 MemCache 节点，可以直接 telnet 过去，并通过 flush_all 让已经存在的键值对立即失效，所以 MemCache 服务器最好配置到内网环境，通过防火墙制定可访问客户端。

6. 不能够遍历 MemCache 中所有的 item。因为这个操作的速度相对缓慢且会阻塞其他的操作。

7. MemCache 的高性能源自于两阶段哈希结构：第一阶段在客户端，通过 Hash 算法根据 Key 值算出一个节点；第二阶段在服务端，通过一个内部的 Hash 算法，查找真正的 item 并返回给客户端。从实现的角度看，MemCache 是一个非阻塞的、基于事件的服务器程序。

8. MemCache 设置添加某一个 Key 值的时候，虽然传入 expiry 为 0 表示这个 Key 值永久有效，但是这个 Key 值也会在 30 天之后失效。由常量 REALTIME_MAXDELTA 60*60*24*30 控制。

9. 最大同时连接数是 200，通过 conn_init()中的 freetotal 进行控制，最大软连接数是 1024，通过 settings.maxconns=1024 进行控制。

10. 跟空间占用相关的参数：settings.factor=1.25, settings.chunk_size=48, 影响 slab 的数据占用和步进方式。

11. MemCache 是键值一一对应，key 默认最大不能超过 128 个字 节，value 默认大小是 1M，也就是一个 slabs，如果要存 2M 的值（连续的），不能用两个 slabs，因为两个 slabs 不是连续的，无法在内存中 存储，故需要修改 slabs 的大小，多个 key 和 value 进行存储时，即使这个 slabs 没有利用完，那么也不会存放别的数据。

12. MemCache 已经可以支持 C/C++、 Perl、 PHP、 Python、 Ruby、 Java、 C#、 Postgres、 Chicken Scheme、 Lua、 MySQL 和 Protocol 等语言客户端。