**memcached命令参数解释**

| 参数 | 参数解释及说明 |
| --- | --- |
| -p <num> | 监听的端口，默认12111 |
| -l <ip\_addr> | 连接的IP地址,，默认是本机。-l选项可以不使用，此时表示在所有网络接口地址上监听。建议是-l <ip\_addr>指定一个内部网络IP地址，以避免成为外部网络攻击的对象 |
| -d start | 启动memcached 服务 |
| -d restart | 重起memcached 服务 |
| -d stop|shutdown | 关闭正在运行的memcached 服务 |
| -d install | 安装memcached 服务 |
| -d uninstall | 卸载memcached 服务 |
| -u <username> | 以<username>的身份运行 (仅在以root运行的时候有效) |
| -m <num> | 最大内存使用，单位MB。默认64MB |
| -M | 内存耗尽时返回错误，而不是删除项 |
| -c <num> | 最大同时连接数，默认是1024 |
| -f <factor> | 块大小增长因子，默认是1.25 |
| -n <bytes> | 最小分配空间，key+value+flags默认是48 |
| -h | 显示帮助 |
| -P | 设置保存Memcached的pid的文件 |
| -vv | 可输出chunk大小及数量 |

使用flush\_all之后，再使用stats items发现key还在，此时只需要使用get获取对应的数据时，才会删除key。

memcached能接受的key的最大长度是多少？

key 的最大长度是250个字符。需要注意的是，250是memcached服务器端内部的限制，如果您使用的客户端支持”key的前缀”或类似特性，那么 key（前缀+原始key）的最大长度是可以超过250个字符的。我们推荐使用使用较短的key，因为可以节省内存和带宽。

memcached对item的过期时间有什么限制？

过期时间最大可以达到30天。memcached把传入的过期时间（时间段）解释成时间点后，一旦到了这个时间点，memcached就把item置为失效状态。这是一个简单但obscure的机制。

memcached最大能存储多大的单个item？

1MB。如果你的数据大于1MB，可以考虑在客户端压缩或拆分到多个key中。

为什么单个item的大小被限制在1M byte之内？

啊…这是一个大家经常问的问题！

简单的回答：因为内存分配器的[**算法**](http://lib.csdn.net/base/datastructure)就是这样的

XMemcached使用：

分布策略：

默认分布的策略是按照key的哈希值模以连接数得到的余数，对应的连接就是将要存储的节点。可设置为“一致性哈希”（consistent hash），或选举散列，代码如下：

|  |
| --- |
| **builder.setSessionLocator(new KetamaMemcachedSessionLocator());**  **builder.setSessionLocator(new ElectionMemcachedSessionLocator());** |

命名空间：

从1.4.2开始，xmemcached提供了memcached命名空间的封装使用，你可以将一组缓存项放到同一个命名空间下，可以让整个命名空间下所有的缓存项同时失效，例子：

|  |
| --- |
| String ns = "namespace" ;  **this**.memcachedClient.withNamespace(ns,  new MemcachedClientCallable<Void>() {  **public** Void call(MemcachedClient client)  throws MemcachedException, InterruptedException,TimeoutException {  //a,b,c都在namespace下  client.**set**("a",1);  client.**set**("b",1);  client.**set**("c",1);  **return** null;  }  });  //获取命名空间内的a对应的值  Integer aValue = **this**.memcachedClient.withNamespace(ns,  new MemcachedClientCallable<Integer>() {  **public** Integer call(MemcachedClient client)  throws MemcachedException, InterruptedException,  TimeoutException {  **return** client.**get**("a");  }  });  //使得命名空间失效  **this**.memcachedClient.invalidateNamespace(ns); |

节点权重：

可在创建MemcachedClientBuilder的时候指定，也可通过XMemcachedClientMBean的setServerWeight()方法来动态指定。

二进制协议：

使用二进制协议的理由是它不需要文本协议的解析处理，使得原本高速的 memcached 的性能更上一 层楼，还能减少文本协议的漏洞

builder.setCommandFactory(new BinaryCommandFactory());

有些方法，如getAndTouch()只能在二进制协议下起作用。

**注：windows版的memcached不支持。**

JMX支持：

可以通过JMX查看xmemcached的状态，在jvm启动参数中添加：

-Dxmemcached.jmx.enable=true

即可通过JMX监控xmemcached状态，xmemcached通过RMI暴露服务接口：

service:jmx:rmi:///jndi/rmi://[host]:7077/xmemcachedServer

你可以在jconsole中查看这些MBean。 提供的MBean包括:

|  |  |
| --- | --- |
| MBean | 描述 |
| net.rubyeye.xmemcached.monitor.StatisticsHandlerMBean | 用于查看Client统计信息 |
| net.rubyeye.xmemcached.impl.OptimizerMBean | 用于调整性能参数 |
| net.rubyeye.xmemcached.XMemcachedClientMBean | 动态添加或者删除节点，查看有效服务器等信息 |

JMX的更多选项:

|  |  |
| --- | --- |
| 选项 | 描述 |
| -Dxmemcached.rmi.port | RMI端口 |
| -Dxmemcached.rmi.name | RMI服务名 |

动态添加/删除节点：

通过client的addServer()和removeServer()方法。

添加节点后，key不会进行重新分布，所以会导致有些key取不到。但是使用**一致性哈希分布策略**，则会尽可能少的导致key取不到。

删除节点后，如果client不进行重新获取，则可能会报错：**Session has been closed**。

NIO连接池：

Xmemcached是基于java nio的client实现，默认对一个memcached节点只有一个连接，这在通常情况下已经有非常优异的表现。但是在典型的高并发环境下,nio的单连接也会遇到性能瓶颈。因此XMemcached支持设置nio的连接池，允许建立多个连接到同一个memcached节点，但是请注意，这些连接之间是不同步的，因此你的应用需要自己保证数据更新的同步，启用连接池可以通过**XMemcachedClientBuilder. setConnectionPoolSize()**方法来实现。

Failure模式和standby节点：

从1.3版本开始，xmemcached支持failure模式。所谓failure模式是指，当一个memcached节点down掉的时候，发往这个节点的请求将直接失败，而不是发送给下一个有效的memcached节点。具体可以看memcached的文档。默认不启用failure模式。可通过**XMemcachedClientBuilder. setFailureMode()**来设置。

xmemcached还支持主辅模式，你可以设置一个memcached的节点的备份节点，当主节点down掉的情况下，会将本来应该发往主节点的请求转发给standby备份节点。使用备份节点的前提是**启用failure模式**。备份节点设置示例如下：

|  |
| --- |
| String ad = “localhost:11211,localhost:11212 host2:11211,host2:11212”;  MemcachedClient builder=new XmemcachedClientBuilder(AddrUtil.getAddressMap(ad)); |

上面的例子，将localhost:11211的备份节点设置为localhost:11212,而将host2:11211的备份节点设置为host2:11212。

数据转换器：

数据中存储到memcached中都是字符串形式的，假如要存储对象类型的数据，则需要进行转换，默认的转换器是序列化转换器SerializingTranscoder。

数据压缩：

memcached存储大数据的效率是比较低的，当数据比较大的时候xmemcached会帮你压缩在存储，取出来的时候自动解压并反序列化，这个大小阈值默认是16K，可以通过Transcoder接口的setCompressionThreshold(1.2.1引入)方法修改阈值。

packZeros：

XMemcached的序列化转换器在序列化数值类型的时候有个特殊处理，如果前面N个字节都是0，那么将会去除这些0，缩减后的数据将更小，例如数字3序列化是0x0003，那么前面3个0将去除掉成一个字节0x3。反序列化的时候将自动在前面根据数值类型补0。这一特性**默认是开启**的，如果考虑到与其他client兼容的话需要可关闭此特性。

sanitizeKeys：

在官方客户端有提供一个sanitizeKeys选项，当选择用URL当key的时候，MemcachedClient会自动将URL encode再存储。默认是**关闭**的，想启用可以通过client.setSanitizeKeys(true)。