NOSQL

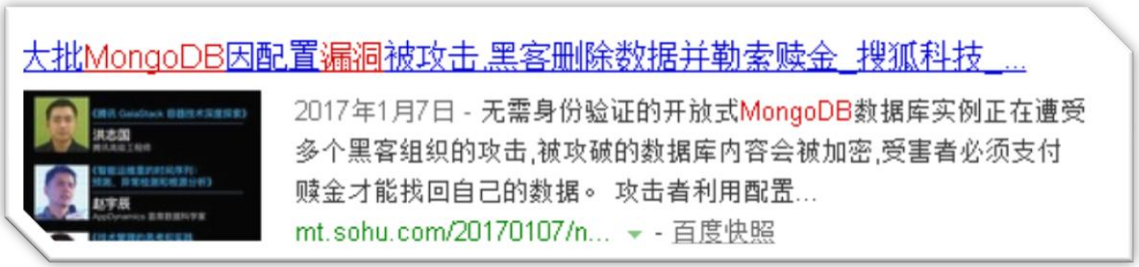
NOSQL 不是“No！SQL!”而是“Not Only SQL”，他不是用来替代关系型数据库的。而是在某些用关系型数据库不合适的地方做优化的。系统中的常规数据仍然用关系型数据库是最合适的。

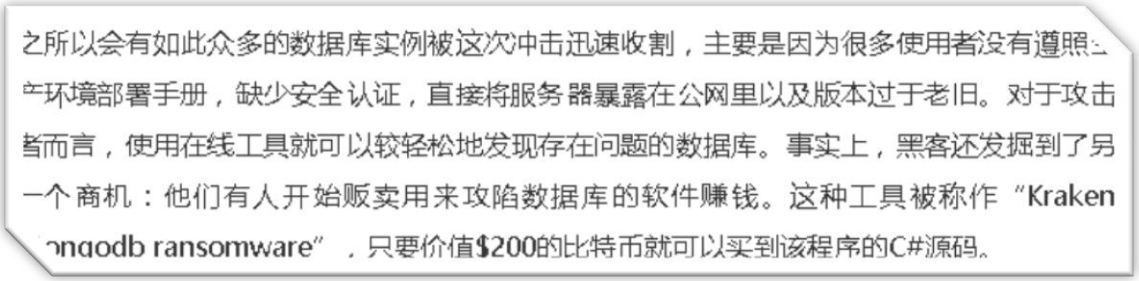
常用的 NoSQL 数据库有 Memcached、Redis、MongoDB 等。Redis、Memcached 属于键值库，MongoDB 属于文档数据库。他们各自都有各自的优缺点、应用场景。

NoSQL 不是关系型数据库，一般没有 ADO.Net 驱动，都是用各自数据库的开发包。

现在 SQLServer、MySQL、Oracle 等数据库中也加入这些 NoSQL 的特性了，但是还是建议“专业的人干专业的事情”。

一定记住，为了安全，NoSQL 数据库、关系数据库等这些永远都不要放到公网，而是放到 web 服务器所在的内网服务器上。





生产环境最好数据库服务器部署到 Linux 下效率是最高的。这些数据库有的默认是允许所有 ip 访问，有的默认是只允许 127.0.0.1 访问。在生产环境最好都配置只允许局域网里某个 ip 才能访问。

Memcached

1、.Net 内置内存缓存

asp.net 中是有缓存的实现：HttpContext.Cache，缓存的数据是放到 Web 服务器的进程内存里。在控制台、WinForm、子线程、SignalR 等不支持 HttpContext 的地方还可以使用 MemoryCache.Default（System.Runtime.Caching 这个程序集中），HttpContext.Cache 其实就是对 MemoryCache 的封装。

写入：MemoryCache.Default.Add("age", 666, DateTimeOffset.Now.AddMinutes(1));

读取：

```
if(MemoryCache.Default.Contains("name"))
{
    int age = (int)MemoryCache.Default["age"];
}
```

进程内缓存最大的优点就是效率高。在可预期数据量不大的情况下推荐使用。

如果数据量比较大或者集群服务器比较多，就要用单独的分布式缓存了，也就是搞一台或者多台专门服务器保存缓存数据，所有服务器都访问分布式缓存服务器。

2、Memcached 简介

Memcached 是一个专门用来做缓存的服务器，而且缓存的数据都在内存中。Memcached 就相当于一个 Dictionary 键值对集合，保存的是键值对，然后根据 key 取 value。

当然 web 服务器和 Memcached 之间还是要网络间通讯，效率还是没有进程内缓存效率高。Memcached 程序重启之后数据就会消失。

3、memcached 的安装

windows 官网只提供了源代码，要自己编译。从如鹏官网下载 memcached-win32-1.4.4-14.zip

解压后，安装成 windows 服务的方法见 INSTALL 文件。

4、.Net 连接 memcached

安装 Memcached 的.Net 开发包：Install-Package EnyimMemcached

1)Memcache 存入的是键值对。Memcache 存入数据的 3 中模式 Set、Replace、Add，根据名字就能猜出来：

Set：存在则覆盖，不存在则新增

Replace：如果存在则覆盖，并且返回 true；如果不存在则不处理，并且返回 false；

Add：如果不存在则新增，并且返回 true；如果存在则不处理，并且返回 false；

没特殊要求一般用 Set 就可以了。

```
MemcachedClientConfiguration mcConfig = new MemcachedClientConfiguration();
mcConfig.AddServer("127.0.0.1:11211");//必须指定端口
using (MemcachedClient client = new MemcachedClient(mcConfig))
{
    client.Store(Enyim.Caching.Memcached.StoreMode.Set, "name", "yzk");
}
```

如果保存普通类对象，则对象必须可序列化（不同 Memcached 客户端保存对象的机制都不尽相同）。

2）存入设置过期时间

设置最后一个 TimeSpan 类型的参数：

client.Store(Enyim.Caching.Memcached.StoreMode.Set, "name", "yzk",TimeSpan.FromSeconds(5));

如果之前对于同一个 Key 设置过一个过期时间，之后又设置过一个，以最后一次的为准。

3)读取：client.Get("name")，如果找不到，则返回 null。当然也可以用 public bool TryGet(string key, out object value)。当然还支持泛型的 public T Get<T>(string key)。

4)Remove(string key)则是删除一个 key 对应的内容。

Key 的长度最高是 250 个字符，Value 最长 1M。

与 Store、Get、Remove 配套的还有 ExecuteXXX 方法，唯一区别就是返回值信息更详细。

5)Key 的选择：

Memcaced 就相当于一个大键值对，不同系统放到 Memcached 中的数据都是不隔离的，因此设定 Key 的时候要选择好 Key，这样就不容易冲突。建议规则“系统名字_模块名字_业务 Key”，比如“Shop_Admin_FilterWords”

6） Increment、Decrement 是用来对计数器进行增减的，不过用得少。用 Redis 更合适。

5、Cas 操作：

用来解决并发问题：读出一个值，做一些判断或者处理，再写回，有可能有并发的问题。

Cas 是 Memcached 1.2.5 之后引入的特性，类似于数据库的“乐观锁”，查询的时候查出一个 cas 值，在写入的时候带着这个 cas 值，如果发现 cas 值已经变了，则说明已经有别人改过了。

下面的程序：

```
var cas = client.GetWithCas("Name");
Console.WriteLine("按任意键继续");
Console.ReadKey();
var res = client.Cas(Enyim.Caching.Memcached.StoreMode.Set, "Name", cas.Result + "1", cas.Cas);
if(res.Result)
{
    Console.WriteLine("修改成功");
}
else
{
    Console.WriteLine("被别人改了");
}
```

启动两个实例，测试效果。

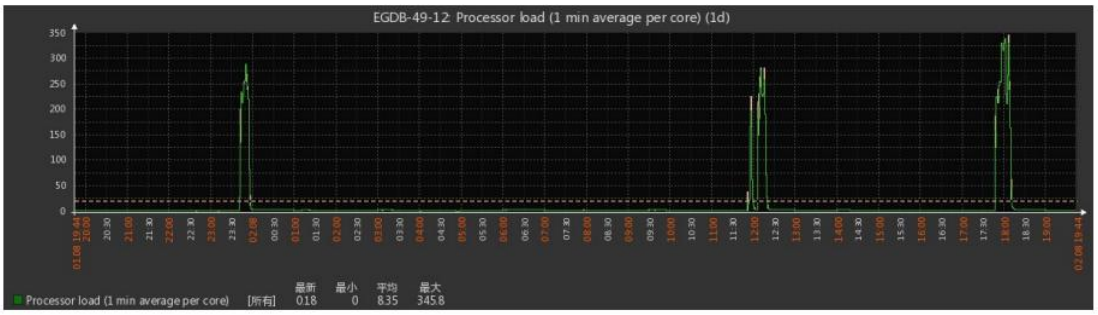
Memcached 一般就是做缓存用，因此也不要用这个 Cas。

6、memcached 的集群

memcached 重启之后短时间内大量的请求会涌入数据库，给数据库造成压力，解决这个的方法就是使用集群，有多台 Memcached 服务器提供服务。

当 memcached 服务器压力大了之后也有必要搞 memcached 集群来分担压力。

Memcached 服务器的“雪崩”问题：如果所有缓存设置过期时间一样，那么每隔一段时间就会造成一次数据库访问的高峰：



解决的方法就是缓存时间设置不一样，比如加上一个随机数。

Memcached 的集群实现很简单，集群节点直接不进行通讯、同步，只要在多个服务器上启动多个 Memcached 服务器即可，客户端决定把数据写入不同的实例，不搞主从复制，每个数据库实例保存一部分内容。

然后 mcConfig.AddServer("127.0.0.1:11211");添加多个服务器 ip 地址，然后客户端根据自己的算法决定把数据写入哪个 Memcached 实例，取数据库的时候再根据同样的定位算法去哪台服务器上去取。

节点定位算法有很多种，最常用的有两种 Ketama、VBucket。Ketama 是根据 Key 算出一个 hash 值，根据 hash 值再算到服务器；而 VBucket 也是根据 key 算出 hash 值，但是不是直接根据 hash 值算出服务地址，而是维护一个 VBucket 表，在表中指定不同的 hash 值由不同的服务器处理，还可以临时改变指向。建议用 Ketama 就可以了。节点定位算法会自动处理故障服务器。mcConfig.NodeLocatorFactory = new KetamaNodeLocatorFactory()。缓存要求都不高。

Redis

1、Redis 简介

Redis 是一个支持数据结构更多的键值对数据库。它的值不仅可以是字符串等基本数据类型，也可以是类对象，更可以是 Set、List、计数器等高级的数据结构。

Memcached 也可以保存类似于 Set、List 这样的结构，但是如果说要向 List 中增加元素，Memcached 则需要把 List 全部元素取出来，然后再把元素增加进去，然后再保存回去，不仅效率低，而且有并发访问问题。Redis 内置的 Set、List 等可以直接支持增加、删除元素的操作，效率很高，操作是原子的。

Memcached 数据存在内存中，memcached 重启后数据就消失；而 Redis 会把数据持久化到硬盘中，Redis 重启后数据还存在。

2、Redis 的安装

redis for windows >=2.8 的版本支持直接安装为 windows 服务

https://github.com/MicrosoftArchive/redis

如果下载 msi 自动装完服务，如果下载 zip 需要按照下面的方法安装为服务：https://raw.githubusercontent.com/MSOpenTech/redis/3.0/Windows%20Service%20Documentation.md

3、redis 的优点：

1) 支持 string、list、set、geo 等复杂的数据结构。
2) 高命中的数据运行时是在内存中，数据最终还是可以保存到磁盘中，这样服务器重启之后数据还在。
3) 服务器是单线程的，来自所有客户端的所有命令都是串行执行的，因此不用担心并发修改（串行操作当然还是有并发问题）的问题，编程模型简单；
4) 支持消息订阅/通知机制，可以用作消息队列；
5) Key、Value 最大长度允许 512M；

4、redis 的缺点：

1) Redis 是单线程的，因此单个 Redis 实例只能使用一个 CPU 核，不能充分发挥服务器的性能。可以在一台服务器上运行多个 Redis 实例，不同实例监听不同端口，再互相组成集群。
2) 做缓存性能不如 Memcached；

5、Memcached 的优点：

1) 多线程，可以充分利用 CPU 多核的性能；
2) 做缓存性能最高；

6、Memcached 的缺点：

1) 只能保存键值对数据，键值对只能是字符串，如果有对象数据只能自己序列化成 json 字符串；
2) 数据保存在内存中，重启后会丢失；
3) Key 最大长度 255 个字符，Value 最长 1M。

7、总结

Memcached 只能当缓存服务器用，也是最合适的；Redis 不仅可以做缓存服务器（性能没有 Memcached 好），还可以存储业务数据。

8、redis 命令行管理客户端：

1）

直接启动 redis 安装目录下的 redis-cli 即可。不用管恶心的自动提示。

执行 set name yzk，就是设置键值对 name=yzk

执行 get name 就是查找名字是 name 的值；

keys *是查找所有的 key

key *n*是查找所有名字中含有 n 的 key

2） 和 Redis 一样，Redis 也是不同系统放到 Redis 中的数据都是不隔离的，因此设定 Key 的时候也要选择好 Key。

3） Redis 服务器默认建了 16 个数据库，Redis 的想法是让大家把不同系统的数据放到不同的数据库中。但是建议大家不要这样用，因为 Redis 是单线程的，不同业务都放到同一个 Redis 实例的话效率就不高，建议放到不同的实例中。因此尽量只用默认的 db0 数据库。

命令行下可以用 select 0、select 1 这样的指令切换数据库，最高为 15。试试在不同数据库下新建、查询数据。

4） 了解的常用的几个命令就可以。所有对数据的操作都可以通过命令行进行，后面讲的.net 操作 Redis 的驱动其实就是对这些命令的封装。

9、redis GUI 管理客户端

GUI 客户端非常多，个人推荐使用 RedisDesktopManager

安装后点击【Connect to Redis Server】连接服务器。

展开节点可以看到所有的 Key，双击 Key 可以查看 Key 的值。在根节点上点右键，选择【Console】，这样就可以输入命令。

10、.net 操作 Redis

用 StackExchange.Redis，而不是 ServiceStack.Redis，因为 StackExchange.Redis 依赖组件少，而且操作更接近原生的 redis 操作，ServiceStack 封装的太厉害，而且有过收费的“前科”。

Install-Package StackExchange.Redis

```
using (ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost:6379"))
{
    IDatabase db = redis.GetDatabase();//默认是访问 db0 数据库，可以通过方法参数指定数字访问不同的数据库
    db.StringSet("Name", "abc");
}
```

支持设置过期时间：db.StringSet("name", "rupeng.com", TimeSpan.FromSeconds(10))

获取数据：string s = db.StringGet("Name")如果查不到则返回 null

Redis 里所有方法几乎都支持异步，比如 StringGetAsync()、StringSetAsync()，尽量用异步方法。

注意看到访问的参数、返回值是 RedisKey、RedisValue 类型，进行了运算符重载，可以和 string、byte[]之间进行隐式转换。

11、Key 操作

Key 操作：因为 Redis 里所有数据类型都是用 KeyValue 保存，因此 Key 操作针对所有数据类型，KeyDelete(RedisKey key)：根据 Key 删除；KeyExists(RedisKey key)判断 Key 是否存在，尽量不要用，因为会有并发问题；KeyExpire(RedisKey key, TimeSpan? expiry)、KeyExpire(RedisKey key, DateTime? expiry)设置过期时间；

12、数据类型

Redis 支持的数据结构：string、list、set、sortedset、hash、geo(redis 3.2 以上版本)。对应的 Redis 客户端里的方法都是 StringXXX、HashXXX、GeoXXX 等方法。不同数据类型的操作方法不能混用，比如不能用 ListXXX 写入的值用 StringXXX 去读取或者写入等操作。

13、String 类型

可以用 StringGet、StringSet 来读写键值对，是基础操作

StringAppend(RedisKey key, RedisValue value)：向 Key 的 Value 中附加内容，不存在则新建；

可以用作计数器：db.StringIncrement("count", 2.5)；给 count 这个计数器增加一个值，如果不存在则从 0 开始加；db.StringDecrement("count",1)计数器减值；获取还是用 StringGet()获取字符串类型的值。比如可以用这个来计算新闻点击量、点赞量，效率非常高。

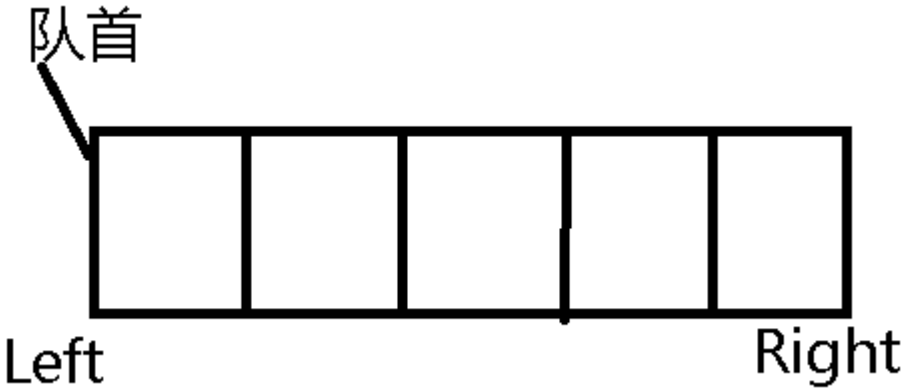
```
private static string XinWen_Prefix = "WWW_XinWen_";
public async Task<ActionResult> Index(int id)
{
    using (ConnectionMultiplexer redis = await ConnectionMultiplexer.ConnectAsync("localhost:6379"))
    {
        IDatabase db = redis.GetDatabase();//默认是访问 db0 数据库，可以通过方法参数指定数字访问不同的数据库
        //以 ip 地址和文章 id 为 key
        string hasClickKey = XinWen_Prefix + Request.UserHostAddress + "_" + id;
```

```
            //如果之前这个 ip 给这个文章贡献过点击量，则不重复计算点击量
            if(await db.KeyExistsAsync(hasClickKey)==false)
            {
                await db.StringIncrementAsync(XinWen_Prefix + "XWClickCount" + id, 1);
                //记录一下这个 ip 给这个文章贡献过点击量，有效期一天
                db.StringSet(hasClickKey, "a", TimeSpan.FromDays(1));
            }

            RedisValue clickCount = await db.StringGetAsync(XinWen_Prefix + "XWClickCount" + id);
            XinWenModel model = new XinWenModel();
            model.ClickCount = Convert.ToInt32(clickCount);
            return View(model);
        }
        return View();
}
```

14、list 类型

Redis 中用 List 保存字符串集合。 比如可以把聊天记录保存到 List 中；商品的物流信息记录。也可以当成双向队列或者双向栈用，list 长度是无限。



ListLeftPush(RedisKey key, RedisValue value)从左侧压栈；RedisValue ListLeftPop(RedisKey key)从左侧弹出；

ListRightPush(RedisKey key, RedisValue value) 从右侧压栈；RedisValue ListRightPop(RedisKey key) 从右侧弹出；

RedisValue ListGetByIndex(RedisKey key, long index)获取 Key 为 key 的 List 中第 index 个元素的值；long ListLength(RedisKey key) 获取 Key 为 key 的 List 中元素个数；尽量不要用 ListGetByIndex、ListLength 因为会有并发问题；。

如果是读取而不 Pop，则使用 ListRange：RedisValue[] ListRange(RedisKey key, long start = 0, long stop = -1)。不传 start、end 表示获取所有数据。指定之后则获取某个范围。

可以把 Redis 的 list 当成消息队列使用，比如向注册用户发送欢迎邮件的工作，可以在注册的流程中把要发送邮件的邮箱放到 list 中，另一个程序从 list 中 pop 获取邮件来发送。

生产者、消费者模式。把生产过程和消费过程隔离。

15、set 类型

如大家所知，set 是一种元素不重复的集合。

SetAdd(RedisKey key, RedisValue value)向 set 中增加元素

bool SetContains(RedisKey key, RedisValue value) 判断 set 中是否存在某个元素；

long SetLength(RedisKey key) 获得 set 中元素的个数；

SetRemove(RedisKey key, RedisValue value)从 set 中删除元素；

RedisValue[] SetMembers(RedisKey key)获取集合中的元素；

如果使用 set 保存封禁用 id 等，就不用做重复性判断了。

注意 set 不是按照插入顺序遍历的，而是按照自己的一个存储方式来遍历，因为没有保存插入的顺序。

16、sortedset

如果对于数据遍历顺序有要求，可以使用 sortedset，他会按照打分来进行遍历。

SortedSetAdd(RedisKey key, RedisValue member, double score) 在 key 这个 sortedset 中增加 member，并且给这个 member 打分，如果 member 已经存在，则覆盖之前的打分；

double SortedSetIncrement(RedisKey key, RedisValue member, double value) 给 key 中 member 这一项增加 value 分；

double SortedSetDecrement(RedisKey key, RedisValue member, double value)：给 key 中 member 这一项减 value 分；

SortedSetEntry[] SortedSetRangeByRankWithScores(RedisKey key, long start = 0, long stop = -1, Order order = Order.Ascending) 根据排序返回 sortedset 中的元素以及元素的打分，start、stop 用来分页查询、order 用来指定排序规则。

测试：

```
db.SortedSetIncrement("Hotwords", "如鹏网", 1);
db.SortedSetIncrement("Hotwords", "如鹏网", 1);
db.SortedSetIncrement("Hotwords", "如鹏网", 1);
db.SortedSetIncrement("Hotwords", "杨中科", 1);
db.SortedSetIncrement("Hotwords", "侯宝林", 1);
db.SortedSetIncrement("Hotwords", "侯宝林", 1);
SortedSetEntry[] items = db.SortedSetRangeByRankWithScores("Hotwords");
foreach(var item in items)
{
    Console.WriteLine(item.Element+"="+item.Score);
}
```

RedisValue[] SortedSetRangeByRank(RedisKey key, long start = 0, long stop = -1, Order order = Order.Ascending) 根据打分排序返回值，可以根据序号查询其中一部分；

RedisValue[] SortedSetRangeByScore(RedisKey key, double start = double.NegativeInfinity, double stop = double.PositiveInfinity, Exclude exclude = Exclude.None, Order order = Order.Ascending, long skip = 0, long take = -1) 根据打分排序返回值，可以只返回 start- stop 这个范围的打分；

sortedset 应用场景：

1） 用户每搜一次一个关键词，就给这个关键词加一分；展示热搜的时候就把前 N 个获取出来就行了；

2） 高积分用户排行榜；

3） 热门商品；

4） 给宝宝投票；

17、Hash

相当于 value 又是一个“键值对集合”或者值是另外一个 Dictionary。

没想到有什么应用场景。

18、Geo 类型

Geo 是 Redis 3.2 版本后新增的数据类型，用来保存兴趣点（POI，point of interest）的坐标信息。可以实现计算两 POI 之间的距离、获取一个点周边指定距离的 POI。

下面添加兴趣点数据，”1”、”2”是点的主键，点的名称、地址、电话等存到其他表中。

```
db.GeoAdd("ShopsGeo", new GeoEntry(116.34039, 39.94218,"1"));
db.GeoAdd("ShopsGeo", new GeoEntry(116.340934, 39.942221, "2"));
db.GeoAdd("ShopsGeo", new GeoEntry(116.341082, 39.941025, "3"));
db.GeoAdd("ShopsGeo", new GeoEntry(116.340848, 39.937758, "4"));
db.GeoAdd("ShopsGeo", new GeoEntry(116.342982, 39.937325, "5"));
db.GeoAdd("ShopsGeo", new GeoEntry(116.340866, 39.936827, "6"));
```

GeoRemove(RedisKey key, RedisValue member)删除一个点

查询两个 POI 之间的举例：double? dist = db.GeoDistance("ShopsGeo", "1", "5", GeoUnit.Meters);//最后一个参数为距离单位

根据点的主键获取坐标：GeoPosition? pos = db.GeoPosition("ShopsGeo", "1")

获取一个 POI 周边的 POI：

```
GeoRadiusResult[] results = db.GeoRadius("ShopsGeo", "2", 200, GeoUnit.Meters);//获取”2”这个周边 200 米范围内的 POI
foreach(GeoRadiusResult result in results)
{
    Console.WriteLine("Id="+result.Member+",位置"+result.Position+"，距离"+result.Distance);
}
```

获取一个坐标（这个坐标不一定是 POI）周边的 POI：

```
GeoRadiusResult[] results = db.GeoRadius("ShopsGeo", 116.34092, 39.94223, 200, GeoUnit.Meters);// 获取(116.34092, 39.94223)这个周边 200 米范围内的 POI
foreach(GeoRadiusResult result in results)
{
    Console.WriteLine("Id="+result.Member+",位置"+result.Position+"，距离"+result.Distance);
}
```

Geo Hash 原理：http://www.cnblogs.com/LBSer/p/3310455.html

19、Redis 的批量操作

如果一次性操作很多，会很慢，那么可以使用批量操作，两种方式：

1）几乎所有的操作都支持数组类型，这样就可以一次性操作多条数据：比如 GeoAdd(RedisKey key, GeoEntry[] values)、SortedSetAdd(RedisKey key, SortedSetEntry[] values)

2）如果一次性的操作不是简单的同类型操作，那么就要使用批量模式：

```
IBatch batch = db.CreateBatch();
db.GeoAdd("ShopsGeo1", new GeoEntry(116.34039, 39.94218, "1"));
db.StringSet("abc", "123");
batch.Execute();
```

会把当前连接的 CreateBatch()、Execute()之间的操作一次性提交给服务器。

20、redis 分布式锁

多线程中的 lock 等的作用范围是当前的程序范围内的，如果想跨多台服务器的锁（尽量避免这样搞），就要使用分布式锁。

```
RedisValue token = Environment.MachineName;
```

```
//实际项目秒杀此处可换成商品 ID
if (db.LockTake("mylock", token, TimeSpan.FromSeconds(10)))//第三个参数为锁超时时间，锁占用最多 10 秒钟，超过 10 秒钟如果还没有 LockRelease，则也自动释放锁，避免了死锁
{
    try
    {
        Console.WriteLine("操作中");
        Thread.Sleep(3000);
        Console.WriteLine("操作完成");
    }
    finally
    {
        db.LockRelease("mylock", token);
    }
}
else
{
    Console.WriteLine("获得锁失败");
}
```

21、抢红包案例

分析 redis 实现抢红包的案例，封上 100 元，随机发给 10 个人。

1) 随机红包的实现：先把 M 元钱平均分配给 N 个人（考虑最后一个除不尽的问题，分完了算 N 个的和，如果多出来一些钱随机发给一个人）；然后执行下面的操作 N 次：生成两个随机的位置 i1、i2，i1 位置的金额为 m1，生成介于[0,m1/2)之间的随机金额 m2，然后从 i1 位置减去这个 m2，再加到 i2 这个位置上。金额有可能是小数，精确到分，因此为了避免精度损失，真实运算的时候都是按照分为单位，只有 int，没有 double。
2) 把这个红包数组以 List 的形式存到 Redis 中；
3) 用户抢红包就是从 List 中 Pop 取红包。

```
string s = "6.66";//红包总金额

int n = 10;//发给几个人

int m = (int)(Convert.ToDouble(s) * 100);//转换为分
int[] bags = new int[n];//n 个红包
int avg = m / n;//算平均值
for(int i=0;i<n;i++)
{
    bags[i] = avg;//先给每个红包平均分配
}

Random rand = new Random();
```

```
int leftM = m - avg * n;//平均分配后可能会剩几分钱，随机发给一个人
bags[rand.Next(0,n)] += leftM;

for(int i=0;i<n;i++)
{
    int i1 = rand.Next(0, n);//随机生成 i1、i2 两个位置
    int i2 = rand.Next(0, n);
    int delta = rand.Next(0, bags[i1]/2);//生成不高于第 i1 个红包目前余额一半的随机数
    bags[i1] -= delta;//从第 i1 个红包减掉 delta 钱
    bags[i2] += delta;//把钱加到第 i2 个红包上
}
if(bags.Sum()!=m)
{
    throw new Exception("红包总金额不符");
}
```

MongoDB

1、 MongoDb 简介

MongoDB 是为互联网而生的数据库，是文档数据库。

MongoDB 的优点：

1) Schema-less，不需要预先定义表结构，同一个“表”中可以保存多个格式的数据；
2) 数据支持嵌套，数据以 json 格式存储；
3) 允许使用 JavaScript 写服务端脚本，类似于存储过程；
4) 支持 Map/Reduce；
5) MongoDB 支持地理位置索引，可以直接用于位置距离计算和查询，实现“附近的人”、“滴滴打车接单”等很容易；

MongoDB 的缺点：

1) Mongodb 没有“数据一致性检查”、“事务”等，不适合存储对数据事务要求高（比如金融）的数据；只适合放非关键性数据（比如日志或者缓存）。
2) 关联查询很弱，不适合做报表查询

2、 MongoDB 服务器的安装

Windows 下安装：

https://www.mongodb.com/ 到 download 的地方下载，下载【Community Server】，里面有各种操作系统下的安装方法。

生产环境肯定要是 Windows-Server，但是在测试学习阶段，如果你电脑是 Win7/8/10，那么下载“Windows Server 2008 64-bit, without SSL support x64”就行，是支持 Win7 以上 64 位系统的。

mongodb 默认使用 C:\data\db\作为数据文件夹，需要**先创建这个文件夹**，然后启动

C:\Program Files\MongoDB\Server\xxx\bin 下的 mongod.exe，这样服务器就启动起来了，如果启动出错一闪而过，那么 cmd 去执行，就能看到报错信息。

把 MongoDB 安装成 windows 服务的方法：

1） 创建一个配置文件 mongod.cfg，在 C:\Program Files\MongoDB\Server\3.4\bin 下，内容是：

```
systemLog:
    destination: file
    path: c:\data\log\mongod.log
storage:
    dbPath: c:\data\db
```

把配置文件中的文件夹创建起来。

2） 注册成系统服务：mongod --config "C:\Program Files\MongoDB\Server\3.4\bin\mongod.cfg" –install

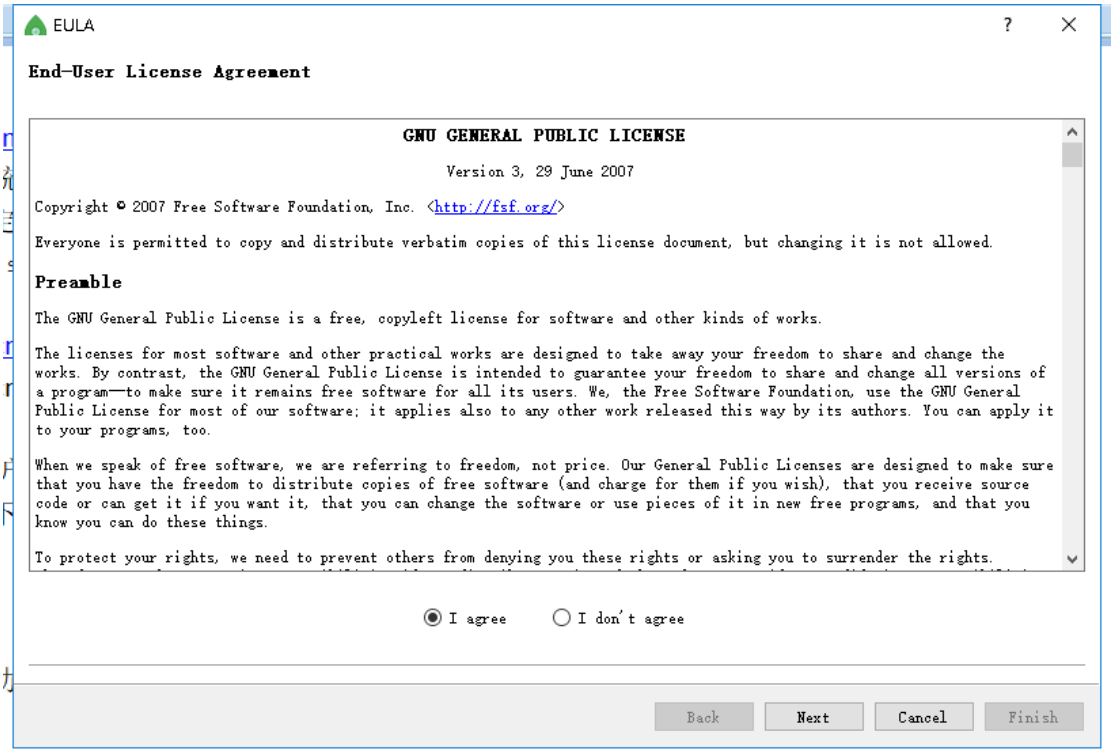
3） 启动服务 `net start MongoDB`

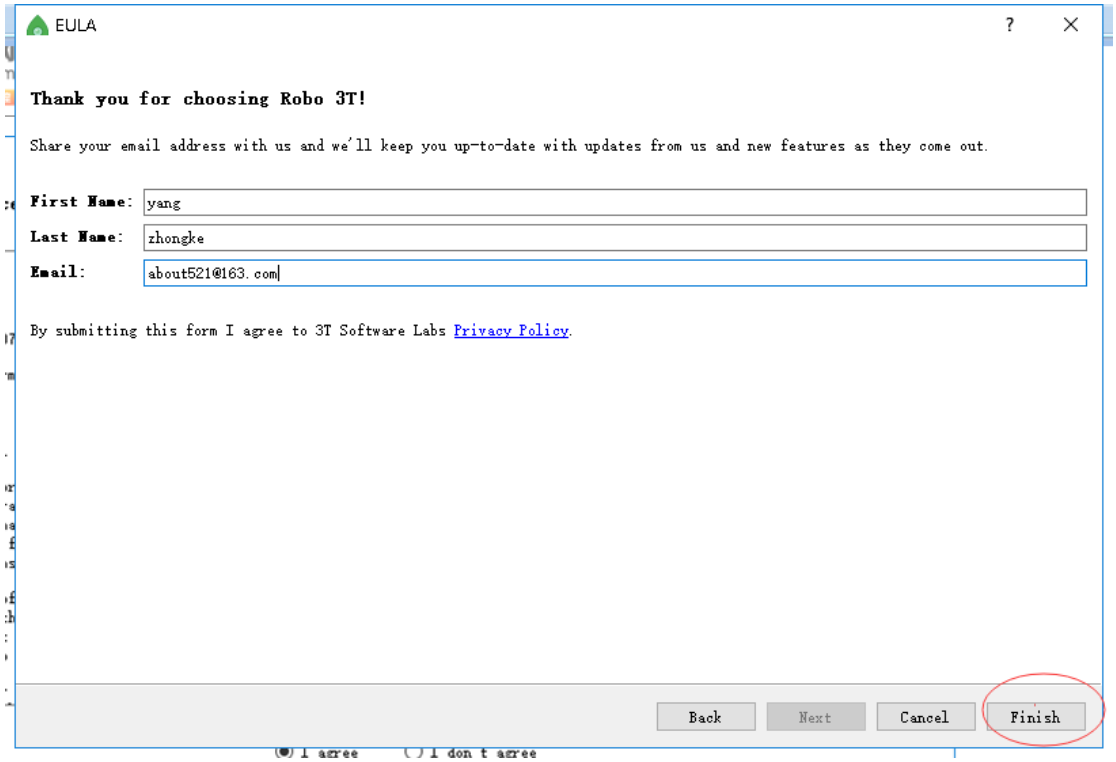
windows 安装方法 https://docs.mongodb.com/master/tutorial/install-mongodb-on-windows/

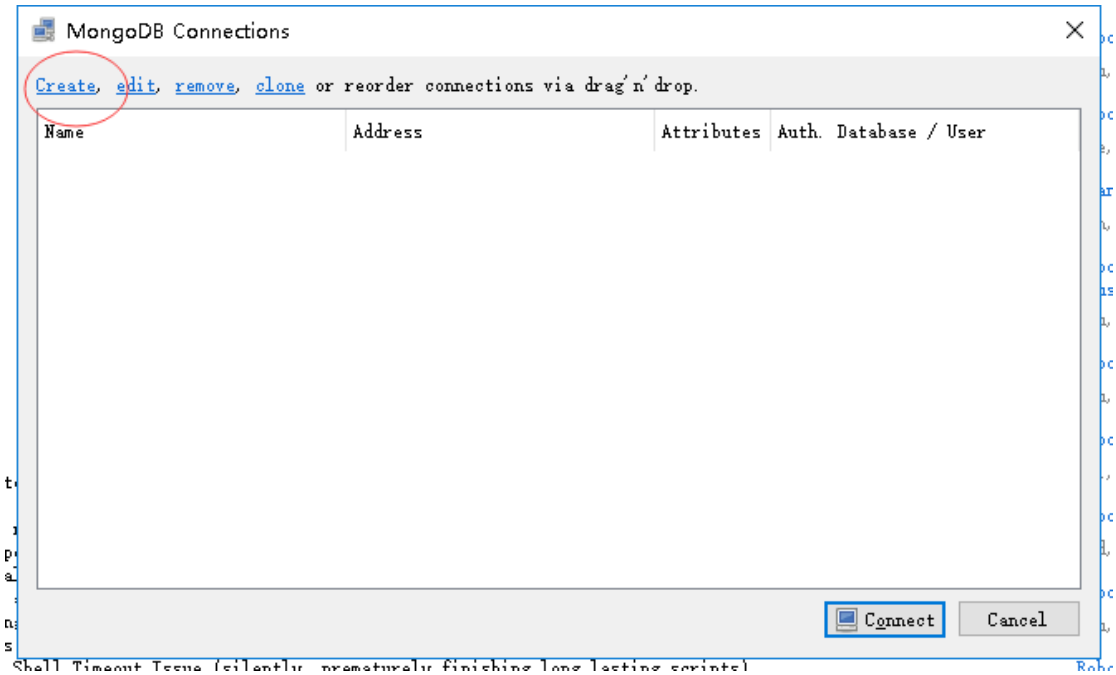
Ubuntu 安装方法 https://docs.mongodb.com/master/tutorial/install-mongodb-on-ubuntu/

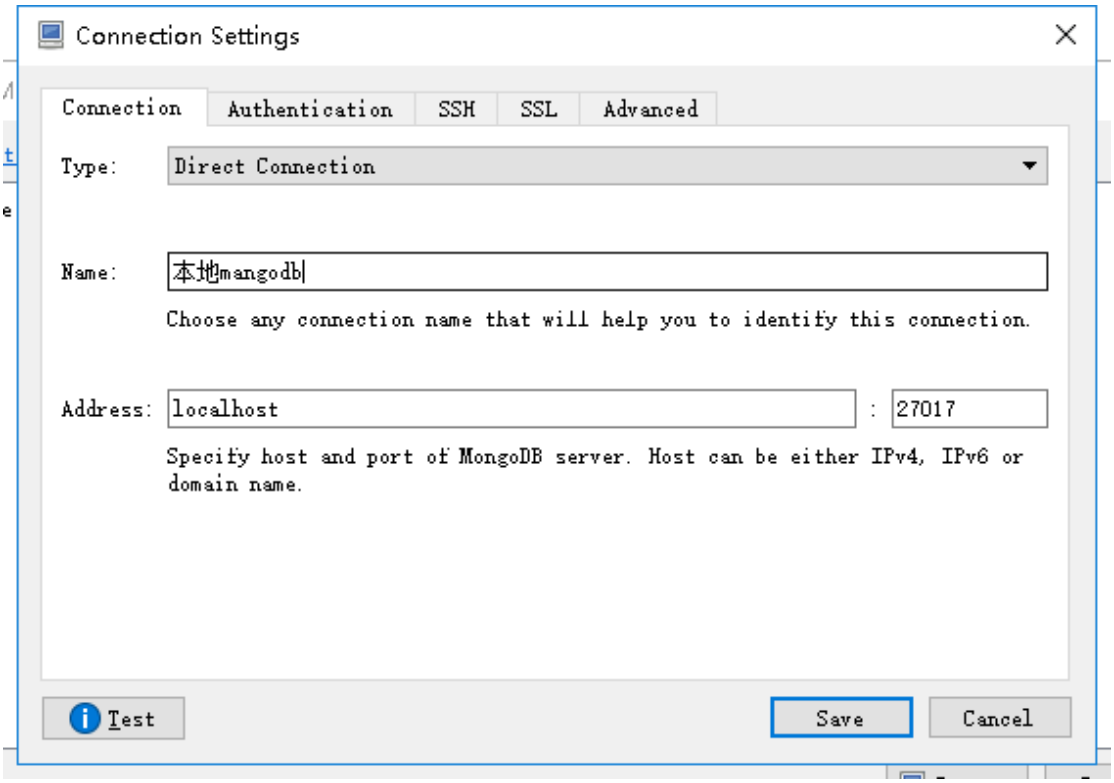
3、 MongoDB GUI 客户端

MongoDB 客户端有很多，有免费的、有收费的，这里推荐一个 Robo 3T，提供下载地址：。。。因为官网的下载地址下载太慢




EULA
Thank you for choosing Robo 3T!
Share your email address with us and we'll keep you up-to-date with updates from us and new features as they come out.
First Name: yang
Last Name: zhongke
Email: about521@163.com
By submitting this form I agree to 3T Software Labs Privacy Policy.
Back
Next
Cancel
Finish


MongoDB Connections
Create, edit, remove, clone or reorder connections via drag'n'drop.
Name
Address
Attributes
Auth. Database / User
Connect
Cancel



4、 .Net 连接 MongoDB

安装.Net 驱动：Install-Package MongoDB.Driver

数据插入：

```
MongoClient client = new MongoClient("mongodb://localhost");
IMongoDatabase database = client.GetDatabase("TestDb1");//相当于数据库
IMongoCollection< Person> collection = database.GetCollection<Person>("Persons");//大致相当于“表”
Person p1 = new Person();
p1.Id = 1;
p1.Name = "rupeng";
p1.Age = 5;
collection.InsertOne (p1);//也支持异步方法，后面建议都用异步的！习惯成自然！
```

回到客户端工具刷新一下，就能看到新插入的数据，MongoDB 会自动创建“数据库”以及 Collection（约等于“表”）。MongoDB 默认用 id 做主键，因此不用显式指定 id 是主键。

MongoDB 中没有内置“自增字段”，可以把 Id 声明为 ObjectId 类型（using MongoDB.Bson）这样插入以后就自动给字段赋值。

```
class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
IMongoCollection<Dog> dogs = database.GetCollection<Dog>("Dogs");
Dog d1 = new Dog();
d1.Age = 33;
d1.Name = "jacky";
dogs.InsertOne(d1);
```

MongoDB 是用 json 保存的，因此也可以直接以 json 格式插入，用 BsonDocument 来代表：

```
IMongoCollection<BsonDocument> dogs = database.GetCollection<BsonDocument>("Dogs");
string json = "{id:8889,Age:81,Name:'japan',gender:true}";
BsonDocument p1 = BsonDocument.Parse(json);
dogs.InsertOne(p1);
```

还可以插入有嵌套关系的对象，比如学生和老师，注意不会有表间关系，都是存到一个集合中，注意和关系库不一样。

5、 查询

```
IMongoCollection<Person> collection = database.GetCollection<Person>("Persons");
var filter1 = Builders<Person>.Filter.Gt(p=>p.Age,5);//Gt：大于。
using (var personsCursor = await collection.FindAsync<Person>(filter1))
{
    while (personsCursor.MoveNext())
    {
        var persons = personsCursor.Current;
        foreach (var p in persons)
        {
            MessageBox.Show(p.Name);
        }
    }
}
```

为什么 FindAsync 不直接返回集合，而是要 MoveNext 之后返回一个集合呢？因为返回的数据量可能很大，因此 MongoDB 是分批下载，下载一批之后执行 GET_More 操作返回下一批。可以通过 FindOptions 参数的 BatchSize 设置每一批的大小。

如果确认返回的数据量不大，可以 var ps = await personsCursor.ToListAsync()（或者 ToEnumerable()等）一下子返回所有数据。还有 Any、First、FirstOrDefault 等以及异步操作。

需要注意 MongoDB 中查询区分大小写。

6、 数据过滤

过滤条件可以写成：var filter1 = Builders<Person>.Filter.Gt("Age",5);或者

除了 Gt，还有 Gte、In、Lt、Lte、Ne、Nin、Near、NearSphere、Or、Where、And、Not。

当然最常用的还是 Where 操作：

```
var filter1 = Builders<Person>.Filter.Where(p => p.Age >= 5 && p.Name == "rupeng");
using (var personsCursor = await collection.FindAsync(filter1))
{
```

```
    foreach (var p in await personsCursor.ToListAsync())
    {
        MessageBox.Show(p.Name);
    }
}
```

7、 分页获取

```
FindOptions<Person,Person> findOpt = new FindOptions<Person, Person>();
findOpt.Limit = 5;//取最多几条
findOpt.Skip = 2;//跳过几条
var filter1 = Builders<Person>.Filter.Where(p => p.Age >= 5 && p.Name == "rupeng");
using (var personsCursor = await collection.FindAsync(filter1, findOpt))
{
    foreach (var p in await personsCursor.ToListAsync())
    {
        MessageBox.Show(p.Name);
    }
}
```

指定排序规则：

```
findOpt.Sort = Builders<Person>.Sort.Ascending(p => p.Age).Descending(p => p.Name);
```

8、 Bson

如果用 BsonDocument，有一些操作还是比较麻烦的：

```
IMongoCollection<BsonDocument> persons = database.GetCollection<BsonDocument>("Persons");
var filter1 = Builders<BsonDocument>.Filter.Gt("Age", 5);
using (var personsCursor = await persons.FindAsync(filter1))
{
    foreach (var p in await personsCursor.ToListAsync())
    {
        MessageBox.Show(p.GetValue("Name").AsString);
    }
}
```

9、 更新数据

```
IMongoCollection<Person> teachers = database.GetCollection<Person>("Persons");
var filter = Builders<Person>.Filter.Where(p => p.Age <= 5);
var update = Builders<Person>.Update
    .Set(p=>p.Age,8);
teachers.UpdateMany(filter, update);
```

10、 删除数据

```
IMongoCollection<Person> teachers = database.GetCollection<Person>("Persons");
var filter = Builders<Person>.Filter.Where(p => p.Age <= 5);
teachers.DeleteMany(filter);
```

用 Update 机会比较少，如果频繁的用 Update 可能意味着用错了；也不要想着 join、group by，还是场景不对！用 MongoDB 做一个分布式日志系统。

11、 MongoDB 应用场景

龙泉寺藏经版本管理系统；日志记录系统；设备监控数据的存储；饿了么外卖骑手接单；存储商品、商家信息；网站评论信息；存储爬虫爬过来的第三方数据；

但是像订单、金融交易、游戏装备等这些关键信息不要用 MongoDB；