**[ASP.NET Redis 开发](http://www.cnblogs.com/jiekzou/p/4487356.html)**

[**文件并发(日志处理)--队列--Redis+Log4Net**](http://www.cnblogs.com/jiekzou/p/4403561.html)

**Redis简介**

Redis是一个开源的，使用C语言编写，面向“键/值”对类型数据的分布式NoSQL数据库系统，特点是高性能，持久存储，适应高并发的应用场景。Redis纯粹为应用而产生，它是一个高性能的key-value数据库,并且提供了多种语言的API

性能测试结果表示SET操作每秒钟可达110000次，GET操作每秒81000次（当然不同的服务器配置性能不同）。

Redis目前提供五种数据类型：string(字符串),list（链表）, Hash（哈希）,set（集合）及zset(sorted set)  （有序集合）

Redis开发维护很活跃，虽然它是一个Key-Value数据库存储系统，但它本身支持MQ功能，所以完全可以当做一个轻量级的队列服务来使用。对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。测试数据分为128Bytes、512Bytes、1K和10K四个不同大小的数据。实验表明：入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则慢的无法忍受；出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于Redis。

**Redis与Memcached的比较.**

1.Memcached是多线程，而Redis使用单线程.

2.Memcached使用预分配的内存池的方式，Redis使用现场申请内存的方式来存储数据，并且可以配置虚拟内存。

3.Redis可以实现持久化，主从复制，实现故障恢复。

4.Memcached只是简单的key与value,但是Redis支持数据类型比较多。

Redis的存储分为内存存储、磁盘存储 .从这一点，也说明了Redis与Memcached是有区别的。Redis 与Memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改 操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

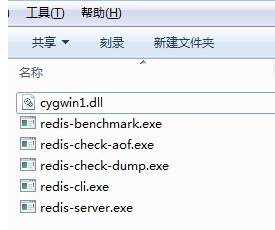
Redis有两种存储方式，默认是snapshot方式，实现方法是定时将内存的快照(snapshot)持久化到硬盘，这种方法缺点是持久化之后如果出现crash则会丢失一段数据。因此在完美主义者的推动下作者增加了aof方式。aof即append only mode，在写入内存数据的同时将操作命令保存到日志文件，在一个并发更改上万的系统中，命令日志是一个非常庞大的数据，管理维护成本非常高，恢复重建时间会非常长，这样导致失去aof高可用性本意。另外更重要的是Redis是一个内存数据结构模型，所有的优势都是建立在对内存复杂数据结构高效的原子操作上，这样就看出aof是一个非常不协调的部分。

其实aof目的主要是数据可靠性及高可用性.

**Redis安装**

文章的最后我提供了下载包，当然你也可以去官网下载最新版本的Redis [https://github.com/dmajkic/redis/downloads](https://github.com/dmajkic/redis/downloads%20)

将服务程序拷贝到一个磁盘上的目录，如下图：



文件说明：

redis-server.exe：服务程序

redis-check-dump.exe：本地数据库检查

redis-check-aof.exe：更新日志检查

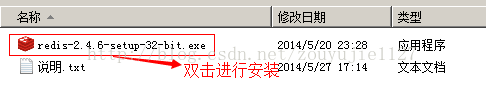
redis-benchmark.exe：性能测试，用以模拟同时由N个客户端发送M个 SETs/GETs 查询.

redis-cli.exe： 服务端开启后，我们的客户端就可以输入各种命令测试了

1、打开一个cmd窗口，使用cd命令切换到指定目录（F:\Redis）运行 redis-server.exe redis.conf

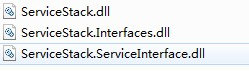
2、重新打开一个cmd窗口，使用cd命令切换到指定目录（F:\Redis）运行 redis-cli.exe -h 127.0.0.1 -p 6379，其中 127.0.0.1是本地ip，6379是redis服务端的默认端口 （这样可以开启一个客户端程序进行特殊指令的测试）.

可以将此服务设置为windows系统服务，下载Redis服务安装软件，安装即可。（https://github.com/rgl/redis/downloads ）

如果你的电脑是64bit系统，可以下载[redis-2.4.6-setup-64-bit.exe](http://pan.baidu.com/s/1qYN4HuW%20" \t "_blank)  


安装完成Redis服务后，我们会在计算机的服务里面看到  
http://img.blog.csdn.net/20141122184241481?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvem91eXVqaWUxMTI3/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

然后启动此服务。

接下来在使用Redis时，还需要下载C#驱动（也就是C#开发库）,如下图：  


**Redis常用数据类型**

使用Redis，我们不用在面对功能单调的数据库时，把精力放在如何把大象放进冰箱这样的问题上，而是利用Redis灵活多变的数据结构和数据操作，为不同的大象构建不同的冰箱。

 Redis最为常用的数据类型主要有以下五种：

* String
* Hash
* List
* Set
* Sorted set

**String类型**

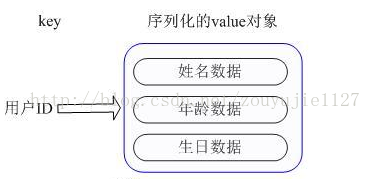
String是最常用的一种数据类型，普通的key/value存储都可以归为此类 。一个Key对应一个Value,string类型是二进制安全的。Redis的string可以包含任何数据，比如jpg图片(生成二进制)或者序列化的对象。基本操作如下：

1. var client = **new RedisClient("127.0.0.1", 6379);**
2. client.Set<**int>("pwd", 1111);**
3. **int pwd=client.Get<int>("pwd");**
4. Console.WriteLine(pwd);
6. UserInfo userInfo = **new UserInfo() { UserName = "zhangsan", UserPwd = "1111" };//</span>(底层使用json序列化 )**
7. client.Set<UserInfo>("userInfo", userInfo);
8. UserInfo user=client.Get<UserInfo>("userInfo");
9. Console.WriteLine(user.UserName);
11. List<UserInfo> list = **new List<UserInfo>() { new UserInfo(){UserName="lisi",UserPwd="111"},new UserInfo(){UserName="wangwu",UserPwd="123"} };**
12. client.Set<List<UserInfo>>("list",list);
13. List<UserInfo>userInfoList=client.Get<List<UserInfo>>("list");
15. **foreach (UserInfo userInfo in userInfoList)**
16. {
17. Console.WriteLine(userInfo.UserName);
18. }

**Hash类型**

Hash是一个string 类型的field和value的映射表。hash特别适合存储对象。相对于将对象的每个字段存成单个string 类型。一个对象存储在hash类型中会占用更少的内存，并且可以更方便的存取整个对象。

作为一个key value存在，很多开发者自然的使用set/get方式来使用Redis，实际上这并不是最优化的使用方法。尤其在未启用VM情况下，Redis全部数据需要放入内存，节约内存尤其重要.



增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回

Redis为单进程单线程模式，采用队列模式将并发访问变为串行访问。Redis本身没有锁的概念，Redis对于多个客户端连接并不存在竞争 .. redis是个单线程的程序，为什么会这么快呢 ?

1、大量线程导致的线程切换开销

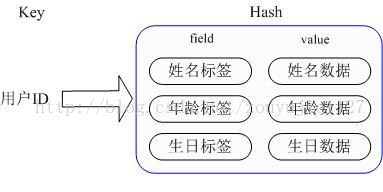
2、锁、

3、非必要的内存拷贝。

4. Redis多样的数据结构，每种结构只做自己爱做的事.

Hash对应的Value内部实际就是一个HashMap,实际这里会有2种不同实现，这个HashMap的成员比较少时,Redis为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的HashMap结构，当成员量增大时会自动转成真正的HashMap.

Key仍然是用户ID, value是一个Map，这个Map的key是成员的属性名，value是属性值，这样对数据的修改和存取都可以直接通过其内部Map的Key(Redis里称内部Map的key为field), 也就是通过 key(用户ID) + field(属性标签) 就可以操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和反序列化



1. client.SetEntryInHash("user", "userInfo", "aaaaaaaaaa");
2. List<**string> list = client.GetHashKeys("user");**
3. List<**string> list = client.GetHashValues("userInfo");//获取值**
4. List<**string> list = client.GetAllKeys();//获取所有的key。**

Redis为不同数据类型分别提供了一组参数来控制内存使用，我们在前面提到过的Redis Hash的value内部是一个

HashMap,如果该Map的成员比较少，则会采用一维数组的方式来紧凑存储该MAP，省去了大量指针的内存开销，这个参数在redis,conf配置文件中下面2项。

Hash-max-zipmap-entries 64

Hash-max-zipmap-value 512.

含义是当value这个Map内部不超过多少个成员时会采用线性紧凑格式存储，默认是64，即value内部有64个以下的成员就是使用线性紧凑存储，超过该值自动转成真正的HashMap.

Hash-max-zipmap-value含义是当value这个MAP内部的每个成员值长度不超过多少字节就会采用线性紧凑存储来节省空间。以上两个条件任意一个条件超过设置值都会转成真正的HashMap，也就不会再节省内存了，这个值设置多少需要权衡，HashMap的优势就是查找和操作时间短。

一个key可对应多个field，一个field对应一个value

这里同时需要注意，Redis提供了接口(hgetall)可以直接取到全部的属性数据,但是如果内部Map的成员很多，那么涉及到遍历整个内部Map的操作，由于Redis单线程模型的缘故，这个遍历操作可能会比较耗时，而另其它客户端的请求完全不响应，这点需要格外注意

建议使用对象类别和ID构成键名，使用字段表示对象属性，字

段值存储属性值，例如：car:2 price 500

**List类型**

list是一个链表结构，主要功能是push,pop,获取一个范围的所有的值等，操作中key理解为链表名字。 Redis的list类型其实就是一个每个子元素都是string类型的双向链表。我们可以通过push,pop操作从链表的头部或者尾部添加删除元素，这样list既可以作为栈，又可以作为队列。Redis list的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构

1. //队列使用
3. client.EnqueueItemOnList("name", "zhangsan");
4. client.EnqueueItemOnList("name", "lisi");
5. **int count= client.GetListCount("name");**
6. **for (int i = 0; i < count; i++)**
7. {
8. Console.WriteLine(client.DequeueItemFromList("name"));
9. }
11. //栈使用
12. client.PushItemToList("name2", "wangwu");
13. client.PushItemToList("name2", "maliu");
14. **int count = client.GetListCount("name2");**
15. **for (int i = 0; i < count; i++)**
16. {
17. Console.WriteLine(client.PopItemFromList("name2"));
18. }

**Set类型**

它是string类型的无序集合。set是通过hash table实现的，添加，删除和查找,对集合我们可以取并集，交集，差集.

1. //对Set类型进行操作
2. client.AddItemToSet("a3", "ddd");
3. client.AddItemToSet("a3", "ccc");
4. client.AddItemToSet("a3", "tttt");
5. client.AddItemToSet("a3", "sssh");
6. client.AddItemToSet("a3", "hhhh");
7. System.Collections.Generic.HashSet<**string>hashset=client.GetAllItemsFromSet("a3");**
8. **foreach (string str in hashset)**
9. {
10. Console.WriteLine(str);
11. }
13. //求并集
14. client.AddItemToSet("a3", "ddd");
15. client.AddItemToSet("a3", "ccc");
16. client.AddItemToSet("a3", "tttt");
17. client.AddItemToSet("a3", "sssh");
18. client.AddItemToSet("a3", "hhhh");
19. client.AddItemToSet("a4", "hhhh");
20. client.AddItemToSet("a4", "h777");
21. System.Collections.Generic.HashSet<**string>hashset=  client.GetUnionFromSets(new string[] { "a3","a4"});**
23. **foreach (string str in hashset)**
24. {
25. Console.WriteLine(str);
26. }
28. //求交集
29. System.Collections.Generic.HashSet<**string> hashset = client.GetIntersectFromSets(new string[] { “a3”, “a4” });**
31. //求差集.
32. System.Collections.Generic.HashSet<**string> hashset = client.GetDifferencesFromSet("a3",new string[] { "a4"});**

返回存在于第一个集合，但是不存在于其他集合的数据。差集

**Sorted Set类型**

sorted set 是set的一个升级版本，它在set的基础上增加了一个顺序的属性，这一属性在添加修改   .元素的时候可以指定，每次指定后，zset(表示有序集合)会自动重新按新的值调整顺序。可以理解为有列的表，一列存 value,一列存顺序。操作中key理解为zset的名字.

Redis sorted set的使用场景与set类似，区别是set不是自动有序的，而sorted set可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。当你需要一个有序的并且不重复的集合列表，那么可以选择sorted set数据结构，

1. client.AddItemToSortedSet("a5", "ffff");
2. client.AddItemToSortedSet("a5","bbbb");
3. client.AddItemToSortedSet("a5", "gggg");
4. client.AddItemToSortedSet("a5", "cccc");
5. client.AddItemToSortedSet("a5", "waaa");
6. System.Collections.Generic.List<**string> list =client.GetAllItemsFromSortedSet("a5");**
7. **foreach (string str in list)**
8. {
9. Console.WriteLine(str);
10. }

**给Redis设置密码**

安装redis服务器后，不能让其在外网环境下面裸奔啊，我们可以为其设置一个复杂的访问密码，最好20位以上，这样可以有效防止别人暴力破解。

找到redis的配置文件，默认在这个地方：C:\Program Files\Redis\conf\redis.conf，查找requirepass选项配置

把这个选项前面的#注释干掉，然后在后面添加一个复杂的密码

# use a very strong password otherwise it will be very easy to break.

#

requirepass 2016@Test.88210\_yujie

# Command renaming.

带密码的访问

var client = new RedisClient("192.168.2.154", 6379); //ip和端口

client.Password = "2016@Test.88210\_yujie"; //redis访问密码

ServiceStack.Redis源码：https://github.com/ServiceStack/ServiceStack.Redis

注：此文使用的redis C#驱动是1.0版本，目前最新版本为4.0，而且已经开源。

4.0版操作Demo如下：

[复制代码](javascript:void(0);)

/\* ==============================================================================

\* 功能描述：ReidsDemo

\* 创 建 者：Zouqj

\* 创建日期：2016/4/14/11:40

\* 更多redis相关技术请参考我的博文：http://www.cnblogs.com/jiekzou/p/4487356.html

==============================================================================\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//添加如下引用

using ServiceStack.Redis;

using ServiceStack.Redis.Generic;

using System.Web.Script.Serialization;

namespace RedisConsoleApplication1

{

// 更多redis相关技术请参考我的博文：http://www.cnblogs.com/jiekzou/p/4487356.html

class Program

{

#region static field

static string host = "192.168.2.154";/\*访问host地址\*/

static string password = "2016@Msd.1127\_kjy";/\*实例id:密码\*/

static readonly RedisClient client = new RedisClient(host, 6379, password);

//static readonly RedisClient client = new RedisClient("xxxxx.m.cnsza.kvstore.aliyuncs.com", 6379, "dacb71347ad0409c:xxxx"); //49正式环境

static IRedisTypedClient<InStoreReceipt> redis = client.As<InStoreReceipt>();

#endregion

static void Main(string[] args)

{

try

{

RedisTestApp();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

#region redis Test

public static void RedisTestApp()

{

StringTest(); //字符串测试

HashTest(); //Hash测试

ObjectTest(); //实体对象测试

SingleObjEnqueueTest(); //单个对象队列测试

ListObjTest(); //对象列表测试

QueueTest(); //队列和栈测试

Console.ReadKey();

}

#region static method

/// <summary>

/// 队列和栈测试

/// </summary>

private static void QueueTest()

{

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*队列 先进先出\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

client.EnqueueItemOnList("test", "饶成龙");//入队。

client.EnqueueItemOnList("test", "周文杰");

long length = client.GetListCount("test");

for (int i = 0; i < length; i++)

{

Console.WriteLine(client.DequeueItemFromList("test"));//出队.

}

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*栈 先进后出\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

client.PushItemToList("name1", "邹琼俊");//入栈

client.PushItemToList("name1", "周文杰");

long length1 = client.GetListCount("name1");

for (int i = 0; i < length1; i++)

{

Console.WriteLine(client.PopItemFromList("name1"));//出栈.

}

Console.ReadKey();

}

/// <summary>

/// 单个对象队列测试

/// </summary>

private static void SingleObjEnqueueTest()

{

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*实体对象队列操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Student \_stu = new Student { Name = "张三", Age = 21 };

JavaScriptSerializer json = new JavaScriptSerializer();

client.EnqueueItemOnList("stu", json.Serialize(\_stu));

\_stu = json.Deserialize<Student>(client.DequeueItemFromList("stu"));

Console.WriteLine(string.Format("姓名：{0},年龄{1}", \_stu.Name, \_stu.Age));

Console.ReadKey();

}

/// <summary>

/// List对象测试

/// </summary>

public static void ListObjTest()

{

List<InStoreReceipt> list = new List<InStoreReceipt>() { new InStoreReceipt() { IdentityID = 1, ReceiptStatus = 1, ReceiptTime = DateTime.Now, ReceiptMessage = "test1" },

new InStoreReceipt() { IdentityID = 2, ReceiptStatus = 1, ReceiptTime = DateTime.Now, ReceiptMessage = "test2" },new InStoreReceipt() { IdentityID = 3, ReceiptStatus = 1, ReceiptTime = DateTime.Now, ReceiptMessage = "test3" }};

AddInStoreInfo(list);

var rList = redis.GetAllItemsFromList(redis.Lists["InStoreReceiptInfoList"]);

rList.ForEach(v => Console.WriteLine(v.IdentityID + "," + v.ReceiptTime + "," + v.ReceiptMessage));

redis.RemoveAllFromList(redis.Lists["InStoreReceiptInfoList"]);

Console.ReadKey();

}

/// <summary>

/// 实体对象测试

/// </summary>

private static void ObjectTest()

{

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*实体对象，单个，列表操作\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

UserInfo userInfo = new UserInfo() { UserName = "zhangsan", UserPwd = "1111" };//</span>(底层使用json序列化 )

client.Set<UserInfo>("userInfo", userInfo);

UserInfo user = client.Get<UserInfo>("userInfo");

Console.WriteLine(user.UserName);

//List<UserInfo> list = new List<UserInfo>() { new UserInfo() { UserName = "lisi", UserPwd = "222" }, new UserInfo() { UserName = "wangwu", UserPwd = "123" } };

//client.Set<List<UserInfo>>("list", list);

List<UserInfo> userInfoList = client.Get<List<UserInfo>>("list");

userInfoList.ForEach(u => Console.WriteLine(u.UserName));

client.Remove("list");

Console.ReadKey();

}

/// <summary>

/// Hash测试

/// </summary>

private static void HashTest()

{

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hash\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

client.SetEntryInHash("userInfoId", "name", "zhangsan");

var lstKeys= client.GetHashKeys("userInfoId");

lstKeys.ForEach(k => Console.WriteLine(k));

var lstValues=client.GetHashValues("userInfoId");

lstValues.ForEach(v => Console.WriteLine(v));

client.Remove("userInfoId");

Console.ReadKey();

}

/// <summary>

/// 字符串测试

/// </summary>

private static void StringTest()

{

#region 字符串类型

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*字符串类型\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

client.Set<string>("name", "zouqj");

string userName = client.Get<string>("name");

Console.WriteLine(userName);

Console.ReadKey();

#endregion

}

/// <summary>

/// 添加需要回执的进仓单信息到Redis

/// </summary>

/// <param name="lstRInStore">进仓单回执信息列表</param>

private static void AddInStoreInfo(List<InStoreReceipt> inStoreReceipt)

{

IRedisList<InStoreReceipt> rlstRInStore = redis.Lists["InStoreReceiptInfoList"];

rlstRInStore.AddRange(inStoreReceipt);

}

#endregion

#endregion

}

/// <summary>

/// 进仓单回执信息（对应清关系统）

/// </summary>

public class InStoreReceipt

{

/// <summary>

/// 主键ID

/// </summary>

public int IdentityID { get; set; }

/// <summary>

/// 回执状态

/// </summary>

public int ReceiptStatus { get; set; }

/// <summary>

/// 回执时间

/// </summary>

public DateTime ReceiptTime { get; set; }

/// <summary>

/// 回执信息

/// </summary>

public string ReceiptMessage { get; set; }

}

public class Student

{

/// <summary>

/// 姓名

/// </summary>

public string Name { get; set; }

/// <summary>

/// 年龄

/// </summary>

public int Age { get; set; }

}

public class UserInfo

{

public string UserName { get; set; }

public string UserPwd { get; set; }

}

}

[复制代码](javascript:void(0);)

RedisHelper类如下：

[复制代码](javascript:void(0);)

public static class RedisHelper

{

#region static field

static string host = "192.168.2.154";/\*访问host地址\*/

static string password = "2016@Msd.1127\_kjy";/\*实例id:密码\*/

static readonly RedisClient client = new RedisClient(host, 6379, password);

#endregion

#region

/// <summary>

/// 获取信息

/// </summary>

/// <typeparam name="T">对象类型</typeparam>

/// <returns>对象</returns>

public static T Get<T>(string token) where T : class,new()

{

return client.Get<T>(token);

}

/// <summary>

/// 设置信息

/// </summary>

/// <typeparam name="T">对象类型</typeparam>

/// <param name="token">key</param>

/// <param name="obj">对象</param>

public static void Set<T>(string token,T obj) where T:class,new()

{

client.Set<T>(token, obj);

}

/// <summary>

/// 设置指定Key的过期时间

/// </summary>

/// <param name="token">具体的key值</param>

/// <param name="seconds">过期时间，单位：秒</param>

public static void Expire(string token,int seconds)

{

client.Expire(token, seconds);

}

#endregion

}

[复制代码](javascript:void(0);)

特别说明：ServiceStack.Redis从4.0开始商用，有每小时6000次访问的限制，如果要在生产环境使用，建议使用低版本。

Redis工具和所需资料代码全下载，地址：<http://pan.baidu.com/s/155F6A>