**ConnectionMultiplexer**

ConnectionMultiplexer 是StackExchange.Redis的核心对象，用这个类的实例来进行Redis的一系列操作，对于一个整个应用程序应该只有一个ConnectionMultiplexer 类的实例。上一章中StackExchangeRedisHelper 的相关代码如下

[复制代码](javascript:void(0);)

private static ConnectionMultiplexer \_instance = null;

/// <summary>

/// 使用一个静态属性来返回已连接的实例，如下列中所示。这样，一旦 ConnectionMultiplexer 断开连接，便可以初始化新的连接实例。

/// </summary>

public static ConnectionMultiplexer Instance

{

get

{

if (\_instance == null)

{

lock (\_locker)

{

if (\_instance == null || !\_instance.IsConnected)

{

\_instance = ConnectionMultiplexer.Connect(Coonstr);

}

}

}

//注册如下事件

\_instance.ConnectionFailed += MuxerConnectionFailed;

\_instance.ConnectionRestored += MuxerConnectionRestored;

\_instance.ErrorMessage += MuxerErrorMessage;

\_instance.ConfigurationChanged += MuxerConfigurationChanged;

\_instance.HashSlotMoved += MuxerHashSlotMoved;

\_instance.InternalError += MuxerInternalError;

return \_instance;

}

}

[复制代码](javascript:void(0);)

**String**

　string类型应该是最长用到的了，用法也很简单，下面展示了用Redis来进行基本的字符串数字存储

[复制代码](javascript:void(0);)

public static IDatabase GetDatabase()

{

return Instance.GetDatabase();

}

/// <summary>

/// 设置缓存

/// </summary>

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

/// <param name="value"></param>

public static void Set(string key, object value, TimeSpan? expiry = default(TimeSpan?), When when = When.Always, CommandFlags flags = CommandFlags.None)

{

key = MergeKey(key);

GetDatabase().StringSet(key, Serialize(value), expiry, when, flags);

}

/// <summary>

/// 根据key获取缓存对象

/// </summary>

/// <typeparam name="T"></typeparam>

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

/// <returns></returns>

public static T Get<T>(string key)

{

key = MergeKey(key);

return Deserialize<T>(GetDatabase().StringGet(key));

}

/// <summary>

/// 移除指定key的缓存

/// </summary>

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

/// <returns></returns>

public static bool Remove(string key)

{

key = MergeKey(key);

return GetDatabase().KeyDelete(key);

}

[复制代码](javascript:void(0);)

 除了基本的string类型操作，Redis同时支持以下几种类型的操作

* **List  列表**
* **Set  无序集合**
* **SortedSet  有序集合**
* **Hash 哈希表**

下面我依次来介绍下这四种类型在StackExchange.Redis中的基本用法

关于代码中的KeyDelete为删除对应的键，我这里是因为测试防止重复才加上的。大家不要误会

**List**

* 特点:有序排列，值可以重复。我们可以通过pop，push操作来从头部和尾部删除或者添加元素。这使得list既可以做栈也可以做队列
* 需求：要求一条微博将最新的10条评论用户名字直接显示在主页上
* 实现：

[复制代码](javascript:void(0);)

public static void LatestUserTop10()

{

IDatabase db = StackExchangeRedisHelper.GetDatabase();

//模拟有一百名用户

for (int i = 1; i <= 100; i++)

{

db.ListLeftPush("user", "用户"+i);

//每一名用户插入后都只保留最后的十个用户到redis数据库中

db.ListTrim("user", 0, 9);

}

RedisValue[] userStores = db.ListRange("user");

foreach (var item in userStores)

{

Console.Write((string)item + ",");

}

db.KeyDelete("user");

Console.ReadLine();

}

[复制代码](javascript:void(0);)

**Set**

* 特点:无序排列，值不可重复。增加删除查询都很快。提供了取并集交集差集等一些有用的操作
* 需求：取两篇文章的评论者的交集并集差集
* 实现：

[复制代码](javascript:void(0);)

public void RedisSetTest()

{

IDatabase db = StackExchangeRedisHelper.GetDatabase();

for (int i = 1; i <= 20; i++)

{

db.SetAdd("文章1", i);

}

for (int i = 15; i <= 35; i++)

{

db.SetAdd("文章2", i);

}

RedisValue[] inter = db.SetCombine(SetOperation.Intersect, "文章1", "文章2");

RedisValue[] union = db.SetCombine(SetOperation.Union, "文章1", "文章2");

RedisValue[] dif1 = db.SetCombine(SetOperation.Difference, "文章1", "文章2");

RedisValue[] dif2 = db.SetCombine(SetOperation.Difference, "文章2", "文章1");

int x = 0;

Console.WriteLine("两篇文章都评论过的用户");

foreach (var item in inter.OrderBy(m => m).ToList())

{

Console.Write((string)item + " ");

}

Console.WriteLine("\n评论过两篇文章中任意一篇文章的用户");

foreach (var item in union.OrderBy(m => m).ToList())

{

Console.Write((string)item + " ");

}

Console.WriteLine("\n只评论过其第一篇文章的用户");

foreach (var item in dif1.OrderBy(m => m).ToList())

{

Console.Write((string)item + " ");

}

Console.WriteLine("\n只评论过其第二篇文章的用户");

foreach (var item in dif2.OrderBy(m => m).ToList())

{

Console.Write((string)item + " ");

}

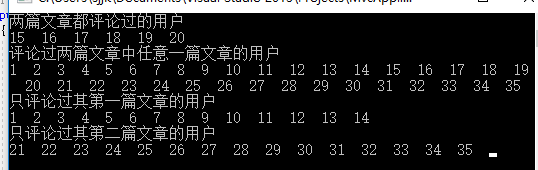
db.KeyDelete("文章1");

db.KeyDelete("文章2");

Console.ReadLine();

}

[复制代码](javascript:void(0);)



**SortedSet**

* 特点:有序排列，值不可重复。类似Set，不同的是sortedset的每个元素都会关联一个double类型的score，用此元素来进行排序
* 需求：显示文章被赞最多的十条评论
* 实现：

[复制代码](javascript:void(0);)

public void HotestUserTop10()

{

IDatabase db = StackExchangeRedisHelper.GetDatabase();

//模拟有一百名评论者，开始每个用户被“赞”的次数为1

List<SortedSetEntry> entrys = new List<SortedSetEntry>();

for (int i = 1; i <= 100; i++)

{

db.SortedSetAdd("文章1", "评论者" + i, 1);

}

//评论者2又被赞了两次

db.SortedSetIncrement("文章1", "评论者2", 2); //对应的值的score+2

//评论者101被赞了4次

db.SortedSetIncrement("文章1", "评论者101", 4); //若不存在该值，则插入一个新的

RedisValue[] userStores = db.SortedSetRangeByRank("文章1", 0, 10, Order.Descending);

for (int i = 0; i < userStores.Length; i++)

{

Console.WriteLine(userStores[i]+":"+ db.SortedSetScore("文章1", userStores[i]));

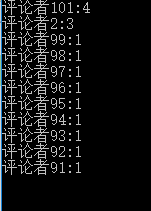
}

db.KeyDelete("文章1");

Console.ReadLine();

}

[复制代码](javascript:void(0);)



**Hash**

* 特点:Hash是一个string类型的field和value的对应表，它更适合来存储对象，相比于每个属性进行一次缓存，利用hash来存储整个对象会占用更小的内存。但是存储速度并不会更快
* 需求：存储一个学生的基本信息
* 实现：

[复制代码](javascript:void(0);)

public void RedisHashTest()

{

IDatabase db = StackExchangeRedisHelper.GetDatabase();

db.HashSet("student1", "name", "张三");

db.HashSet("student1", "age", 12);

db.HashSet("student1", "class", "五年级");

Console.WriteLine(db.HashGet("student1", "name"));

RedisValue[] result = db.HashGet("student1", new RedisValue[] { "name", "age","class" });

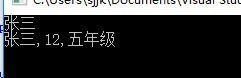
Console.WriteLine(string.Join(",",result));

db.KeyDelete("student1");

Console.ReadLine();

}

[复制代码](javascript:void(0);)



 以下代码是我分别用stringset和hash来存储对象进行的时间及内存比较，内存可通过redis的info命令来查看。

最终显示耗时方面stringset稍微快一点点，内存占用stringset却是hash的二倍

[复制代码](javascript:void(0);)

public void RedisHashVsStringSet()

{

IDatabase db = StackExchangeRedisHelper.GetDatabase();

Stopwatch sw = new Stopwatch();

sw.Start();

//for (int i = 0; i < 100000; i++)

//{

// db.HashSet("studenths" + i, "name", "张三" + i);

// db.HashSet("studenths" + i, "age", 12 + i);

// db.HashSet("studenths" + i, "class", "五年级" + i);

//}

//Console.WriteLine(sw.Elapsed.TotalMilliseconds);

//sw.Restart();

for (int i = 0; i < 100000; i++)

{

db.StringSet("studentstr\_name" + i, "张三" + i);

db.StringSet("studentstr\_age" + i, 12 + i);

db.StringSet("studentstr\_class" + i, "五年级" + i);

}

Console.WriteLine(sw.Elapsed.TotalMilliseconds);

//for (int i = 0; i < 100000; i++)

//{

// db.KeyDelete("studenths" + i);

// db.KeyDelete("studentstr\_name" + i);

// db.KeyDelete("studentstr\_age" + i);

// db.KeyDelete("studentstr\_class" + i);

//}

Console.ReadLine();

}