**[分布式中使用Redis实现Session共享](http://www.cnblogs.com/yanweidie/p/4719692.html)**

　　上一篇介绍了如何使用[nginx+iis](http://www.cnblogs.com/yanweidie/p/4658136.html" \t "_blank)部署一个简单的分布式系统，文章结尾留下了几个问题，其中一个是"如何解决多站点下Session共享"。这篇文章将会介绍如何使用Redis，下一篇在此基础上实现Session。

　　这里特别说明一下，其实没有必要使用Redis来解决Session共享。Asp.net提供了StateServer模式来共享Session，这里重复造轮子的目的1:熟悉Redis的基本知识和使用 2.学习和巩固Session的实现原理。

　　3.学习Redis应用场景

**阅读目录**

* [Redis安装配置](http://www.cnblogs.com/yanweidie/p/4719692.html#_label0)
* [五种数据类型使用](http://www.cnblogs.com/yanweidie/p/4719692.html#_label1)
* [封装拓展](http://www.cnblogs.com/yanweidie/p/4719692.html#_label2)
* [总结](http://www.cnblogs.com/yanweidie/p/4719692.html#_label3)

**Redis安装配置**

**redis**是一个key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。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。

　 最新版本的redis版本为3.0.3，支持集群功能。我这下载的是window版本的，实际场景都是安装在linux系统下的。下载地址：[redis-2.8.19.rar](http://files.cnblogs.com/files/yanweidie/redis-2.8.19.rar" \t "_blank) 。更多下载地址：

    官网 ：<http://redis.io/download>  MSOpenTech：<https://github.com/MSOpenTech/redis>  dmajkic：<https://github.com/dmajkic/redis/downloads>

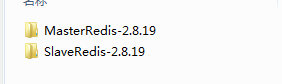
    下载完成之后解压运行redis-server.exe就启动了redis了，启动后会在进程里面看到reids。



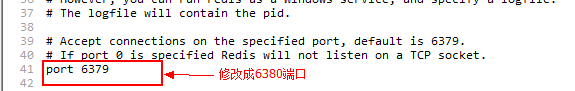
**1.读写分离配置**

   redis的读写分离需要修改配置文件，把解压的文件复制了一份。两份文件是一样的，分别命名为MasterRedis-2.8.19（主redis服务）,SlaveRedis-2.8.19（从redis服务）。redis默认绑定的是6379端口,

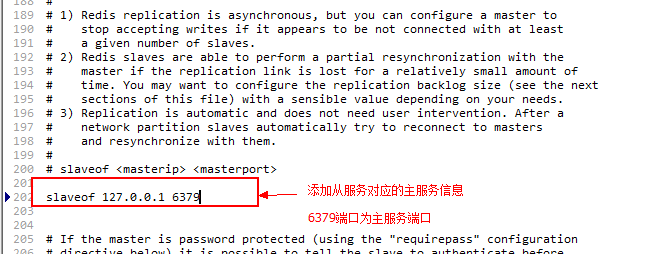
　我们保持主服务配置不变，修改从服务配置。



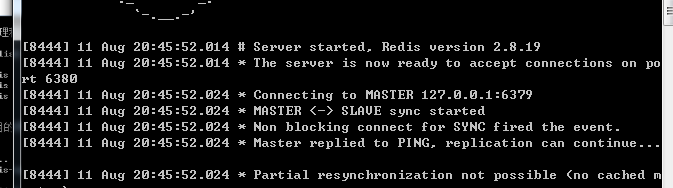
* 修改从服务绑定端口（修改时可以直接搜索port关键字）



* 修改从服务对应的主服务地址（修改时可以直接搜索slaveof关键字）



* 配置文件修改完成以后，分别启动主服务和从服务

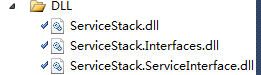


     从服务启动以后，主服务会发送一条同步的sync命令，同步从服务器的缓存数据。

[回到顶部](http://www.cnblogs.com/yanweidie/p/4719692.html#_labelTop)

**五种数据类型使用**

   服务搭建好以后可以使用.net版本redis操作类库[ServiceStack.Redis](https://github.com/ServiceStack/ServiceStack.Redis)来操作redis，本文会用到以下三个dll。



  初始化RedisClient对象

var client = new RedisClient("120.26.197.185", 6379);

**1.String**

    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，value其实不仅是String，也可以是数字：比如想知道什么时候封锁一个IP地址(访问超过几次)。INCRBY命令让这些变得很容易，通过原子递增保持计数。

[复制代码](javascript:void(0);)#region "字符串类型"

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

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

Console.WriteLine(userName);

//访问次数

client.Set<int>("IpAccessCount", 0);

//次数递增

client.Incr("IpAccessCount");

Console.WriteLine(client.Get<int>("IpAccessCount"));

#endregion

[复制代码](javascript:void(0);)

http://images0.cnblogs.com/blog2015/336693/201508/112105161762404.png

**2.Hash**

一个hashid可以存储多项信息，每一项信息也有自己的key

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

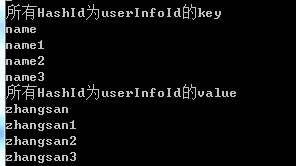
client.SetEntryInHash("userInfoId", "name1", "zhangsan1");

client.SetEntryInHash("userInfoId", "name2", "zhangsan2");

client.SetEntryInHash("userInfoId", "name3", "zhangsan3");

client.GetHashKeys("userInfoId").ForEach(e => Console.WriteLine(e));

client.GetHashValues("userInfoId").ForEach(e => Console.WriteLine(e));



**3.List**

　　应用场景： 

* Redis list的应用场景非常多，也是Redis最重要的数据结构之一。
* 我们可以轻松地实现最新消息排行等功能。
* Lists的另一个应用就是消息队列，可以利用Lists的PUSH操作，将任务存在Lists中，然后工作线程再用POP操作将任务取出进行执行。

[复制代码](javascript:void(0);)

#region "List类型"

client.AddItemToList("userInfoId1", "123");

client.AddItemToList("userInfoId1", "1234");

Console.WriteLine("List数据项条数:" + client.GetListCount("userInfoId1"));

Console.WriteLine("List数据项第一条数据:" + client.GetItemFromList("userInfoId1", 0));

Console.WriteLine("List所有数据");

client.GetAllItemsFromList("userInfoId1").ForEach(e => Console.WriteLine(e));

#endregion

#region "List类型做为队列和栈使用"

Console.WriteLine(client.GetListCount("userInfoId1"));

//队列先进先出

//Console.WriteLine(client.DequeueItemFromList("userInfoId1"));

//Console.WriteLine(client.DequeueItemFromList("userInfoId1"));

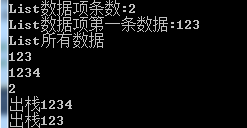
//栈后进先出

Console.WriteLine("出栈"+client.PopItemFromList("userInfoId1"));

Console.WriteLine("出栈"+client.PopItemFromList("userInfoId1"));

#endregion

[复制代码](javascript:void(0);)



**4.Set**

　　应用场景：

* Redis set对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。
* 比如在微博应用中，每个人的好友存在一个集合（set）中，这样求两个人的共同好友的操作，可能就只需要用求交集命令即可。
* Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实

[复制代码](javascript:void(0);)

client.AddItemToSet("A", "B");

client.AddItemToSet("A", "C");

client.AddItemToSet("A", "D");

client.AddItemToSet("A", "E");

client.AddItemToSet("A", "F");

client.AddItemToSet("B", "C");

client.AddItemToSet("B", "F");

//求差集

Console.WriteLine("A,B集合差集");

client.GetDifferencesFromSet("A", "B").ToList<string>().ForEach(e => Console.Write(e + ","));

//求集合交集

Console.WriteLine("\nA,B集合交集");

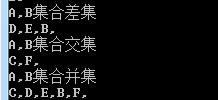
client.GetIntersectFromSets(new string[] { "A", "B" }).ToList<string>().ForEach(e => Console.Write(e + ","));

//求集合并集

Console.WriteLine("\nA,B集合并集");

client.GetUnionFromSets(new string[] { "A", "B" }).ToList<string>().ForEach(e => Console.Write(e + ","));

[复制代码](javascript:void(0);)



**5.Sort Set(排序)**

　　应用场景：

* 以某个条件为权重，比如按顶的次数排序.
* ZREVRANGE命令可以用来按照得分来获取前100名的用户，ZRANK可以用来获取用户排名，非常直接而且操作容易。
* 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。
* 比如:twitter 的public timeline可以以发表时间作为score来存储，这样获取时就是自动按时间排好序的。
* 比如:全班同学成绩的SortedSets，value可以是同学的学号，而score就可以是其考试得分，这样数据插入集合的，就已经进行了天然的排序。
* 另外还可以用Sorted Sets来做带权重的队列，比如普通消息的score为1，重要消息的score为2，然后工作线程可以选择按score的倒序来获取工作任务。让重要的任务优先执行。

[复制代码](javascript:void(0);)

#region "有序Set操作"

client.AddItemToSortedSet("SA", "B", 2);

client.AddItemToSortedSet("SA", "C", 1);

client.AddItemToSortedSet("SA", "D", 5);

client.AddItemToSortedSet("SA", "E", 3);

client.AddItemToSortedSet("SA", "F", 4);

//有序集合降序排列

Console.WriteLine("\n有序集合降序排列");

client.GetAllItemsFromSortedSetDesc("SA").ForEach(e => Console.Write(e + ","));

Console.WriteLine("\n有序集合升序序排列");

client.GetAllItemsFromSortedSet("SA").ForEach(e => Console.Write(e + ","));

client.AddItemToSortedSet("SB", "C", 2);

client.AddItemToSortedSet("SB", "F", 1);

client.AddItemToSortedSet("SB", "D", 3);

Console.WriteLine("\n获得某个值在有序集合中的排名，按分数的升序排列");

Console.WriteLine(client.GetItemIndexInSortedSet("SB", "D"));

Console.WriteLine("\n获得有序集合中某个值得分数");

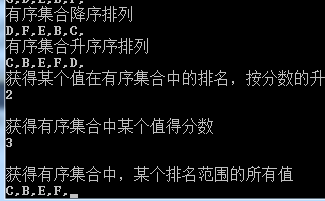
Console.WriteLine(client.GetItemScoreInSortedSet("SB", "D"));

Console.WriteLine("\n获得有序集合中，某个排名范围的所有值");

client.GetRangeFromSortedSet("SA", 0, 3).ForEach(e => Console.Write(e + ","));

#endregion

[复制代码](javascript:void(0);)



**封装拓展**

　　最后提供一份别人写好的Redis操作的帮助类，用到了PooledRedisClientManager连接池来获取RedisClient，同时用到了读写分离的概念，可以直接拿来使用

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using ServiceStack.Redis;

namespace Com.Redis

{

/// <summary>

/// 来源：http://blog.wx6.org/2013/349.htm

/// </summary>

public class RedisBase

{

private static string[] ReadWriteHosts = System.Configuration.ConfigurationSettings.AppSettings["readWriteHosts"].Split(new char[] { ';' });

private static string[] ReadOnlyHosts = System.Configuration.ConfigurationSettings.AppSettings["readOnlyHosts"].Split(new char[] { ';' });

#region -- 连接信息 --

public static PooledRedisClientManager prcm = CreateManager(ReadWriteHosts, ReadOnlyHosts);

private static PooledRedisClientManager CreateManager(string[] readWriteHosts, string[] readOnlyHosts)

{

// 支持读写分离，均衡负载

return new PooledRedisClientManager(readWriteHosts, readOnlyHosts, new RedisClientManagerConfig

{

MaxWritePoolSize = 5, // “写”链接池链接数

MaxReadPoolSize = 5, // “读”链接池链接数

AutoStart = true,

});

}

#endregion

#region -- Item --

/// <summary>

/// 设置单体

/// </summary>

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

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

/// <param name="t"></param>

/// <param name="timeSpan"></param>

/// <returns></returns>

public static bool Item\_Set<T>(string key, T t)

{

try

{

using (IRedisClient redis = prcm.GetClient())

{

return redis.Set<T>(key, t, new TimeSpan(1, 0, 0));

}

}

catch (Exception ex)

{

// LogInfo

}

return false;

}

/// <summary>

/// 获取单体

/// </summary>

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

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

/// <returns></returns>

public static T Item\_Get<T>(string key) where T : class

{

using (IRedisClient redis = prcm.GetClient())

{

return redis.Get<T>(key);

}

}

/// <summary>

/// 移除单体

/// </summary>

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

public static bool Item\_Remove(string key)

{

using (IRedisClient redis = prcm.GetClient())

{

return redis.Remove(key);

}

}

#endregion

#region -- List --

public static void List\_Add<T>(string key, T t)

{

using (IRedisClient redis = prcm.GetClient())

{

var redisTypedClient = redis.GetTypedClient<T>();

redisTypedClient.AddItemToList(redisTypedClient.Lists[key], t);

}

}

public static bool List\_Remove<T>(string key, T t)

{

using (IRedisClient redis = prcm.GetClient())

{

var redisTypedClient = redis.GetTypedClient<T>();

return redisTypedClient.RemoveItemFromList(redisTypedClient.Lists[key], t) > 0;

}

}

public static void List\_RemoveAll<T>(string key)

{

using (IRedisClient redis = prcm.GetClient())

{

var redisTypedClient = redis.GetTypedClient<T>();

redisTypedClient.Lists[key].RemoveAll();

}

}

public static int List\_Count(string key)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

return redis.GetListCount(key);

}

}

public static List<T> List\_GetRange<T>(string key, int start, int count)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

var c = redis.GetTypedClient<T>();

return c.Lists[key].GetRange(start, start + count - 1);

}

}

public static List<T> List\_GetList<T>(string key)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

var c = redis.GetTypedClient<T>();

return c.Lists[key].GetRange(0, c.Lists[key].Count);

}

}

public static List<T> List\_GetList<T>(string key, int pageIndex, int pageSize)

{

int start = pageSize \* (pageIndex - 1);

return List\_GetRange<T>(key, start, pageSize);

}

/// <summary>

/// 设置缓存过期

/// </summary>

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

/// <param name="datetime"></param>

public static void List\_SetExpire(string key, DateTime datetime)

{

using (IRedisClient redis = prcm.GetClient())

{

redis.ExpireEntryAt(key, datetime);

}

}

#endregion

#region -- Set --

public static void Set\_Add<T>(string key, T t)

{

using (IRedisClient redis = prcm.GetClient())

{

var redisTypedClient = redis.GetTypedClient<T>();

redisTypedClient.Sets[key].Add(t);

}

}

public static bool Set\_Contains<T>(string key, T t)

{

using (IRedisClient redis = prcm.GetClient())

{

var redisTypedClient = redis.GetTypedClient<T>();

return redisTypedClient.Sets[key].Contains(t);

}

}

public static bool Set\_Remove<T>(string key, T t)

{

using (IRedisClient redis = prcm.GetClient())

{

var redisTypedClient = redis.GetTypedClient<T>();

return redisTypedClient.Sets[key].Remove(t);

}

}

#endregion

#region -- Hash --

/// <summary>

/// 判断某个数据是否已经被缓存

/// </summary>

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

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

/// <param name="dataKey"></param>

/// <returns></returns>

public static bool Hash\_Exist<T>(string key, string dataKey)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

return redis.HashContainsEntry(key, dataKey);

}

}

/// <summary>

/// 存储数据到hash表

/// </summary>

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

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

/// <param name="dataKey"></param>

/// <returns></returns>

public static bool Hash\_Set<T>(string key, string dataKey, T t)

{

using (IRedisClient redis = prcm.GetClient())

{

string value = ServiceStack.Text.JsonSerializer.SerializeToString<T>(t);

return redis.SetEntryInHash(key, dataKey, value);

}

}

/// <summary>

/// 移除hash中的某值

/// </summary>

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

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

/// <param name="dataKey"></param>

/// <returns></returns>

public static bool Hash\_Remove(string key, string dataKey)

{

using (IRedisClient redis = prcm.GetClient())

{

return redis.RemoveEntryFromHash(key, dataKey);

}

}

/// <summary>

/// 移除整个hash

/// </summary>

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

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

/// <param name="dataKey"></param>

/// <returns></returns>

public static bool Hash\_Remove(string key)

{

using (IRedisClient redis = prcm.GetClient())

{

return redis.Remove(key);

}

}

/// <summary>

/// 从hash表获取数据

/// </summary>

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

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

/// <param name="dataKey"></param>

/// <returns></returns>

public static T Hash\_Get<T>(string key, string dataKey)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

string value = redis.GetValueFromHash(key, dataKey);

return ServiceStack.Text.JsonSerializer.DeserializeFromString<T>(value);

}

}

/// <summary>

/// 获取整个hash的数据

/// </summary>

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

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

/// <returns></returns>

public static List<T> Hash\_GetAll<T>(string key)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

var list = redis.GetHashValues(key);

if (list != null && list.Count > 0)

{

List<T> result = new List<T>();

foreach (var item in list)

{

var value = ServiceStack.Text.JsonSerializer.DeserializeFromString<T>(item);

result.Add(value);

}

return result;

}

return null;

}

}

/// <summary>

/// 设置缓存过期

/// </summary>

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

/// <param name="datetime"></param>

public static void Hash\_SetExpire(string key, DateTime datetime)

{

using (IRedisClient redis = prcm.GetClient())

{

redis.ExpireEntryAt(key, datetime);

}

}

#endregion

#region -- SortedSet --

/// <summary>

/// 添加数据到 SortedSet

/// </summary>

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

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

/// <param name="t"></param>

/// <param name="score"></param>

public static bool SortedSet\_Add<T>(string key, T t, double score)

{

using (IRedisClient redis = prcm.GetClient())

{

string value = ServiceStack.Text.JsonSerializer.SerializeToString<T>(t);

return redis.AddItemToSortedSet(key, value, score);

}

}

/// <summary>

/// 移除数据从SortedSet

/// </summary>

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

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

/// <param name="t"></param>

/// <returns></returns>

public static bool SortedSet\_Remove<T>(string key, T t)

{

using (IRedisClient redis = prcm.GetClient())

{

string value = ServiceStack.Text.JsonSerializer.SerializeToString<T>(t);

return redis.RemoveItemFromSortedSet(key, value);

}

}

/// <summary>

/// 修剪SortedSet

/// </summary>

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

/// <param name="size">保留的条数</param>

/// <returns></returns>

public static int SortedSet\_Trim(string key, int size)

{

using (IRedisClient redis = prcm.GetClient())

{

return redis.RemoveRangeFromSortedSet(key, size, 9999999);

}

}

/// <summary>

/// 获取SortedSet的长度

/// </summary>

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

/// <returns></returns>

public static int SortedSet\_Count(string key)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

return redis.GetSortedSetCount(key);

}

}

/// <summary>

/// 获取SortedSet的分页数据

/// </summary>

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

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

/// <param name="pageIndex"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

public static List<T> SortedSet\_GetList<T>(string key, int pageIndex, int pageSize)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

var list = redis.GetRangeFromSortedSet(key, (pageIndex - 1) \* pageSize, pageIndex \* pageSize - 1);

if (list != null && list.Count > 0)

{

List<T> result = new List<T>();

foreach (var item in list)

{

var data = ServiceStack.Text.JsonSerializer.DeserializeFromString<T>(item);

result.Add(data);

}

return result;

}

}

return null;

}

/// <summary>

/// 获取SortedSet的全部数据

/// </summary>

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

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

/// <param name="pageIndex"></param>

/// <param name="pageSize"></param>

/// <returns></returns>

public static List<T> SortedSet\_GetListALL<T>(string key)

{

using (IRedisClient redis = prcm.GetReadOnlyClient())

{

var list = redis.GetRangeFromSortedSet(key, 0, 9999999);

if (list != null && list.Count > 0)

{

List<T> result = new List<T>();

foreach (var item in list)

{

var data = ServiceStack.Text.JsonSerializer.DeserializeFromString<T>(item);

result.Add(data);

}

return result;

}

}

return null;

}

/// <summary>

/// 设置缓存过期

/// </summary>

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

/// <param name="datetime"></param>

public static void SortedSet\_SetExpire(string key, DateTime datetime)

{

using (IRedisClient redis = prcm.GetClient())

{

redis.ExpireEntryAt(key, datetime);

}

}

#endregion

}

}

[复制代码](javascript:void(0);)

View Code

　使用很简单，几行代码

//会往主服务里面写入

RedisBase.Hash\_Set<string>("PooledRedisClientManager", "one", "123");

//从服务里面读取信息

RedisBase.Hash\_Get<string>("PooledRedisClientManager", "one");

   连接池的初始化

[复制代码](javascript:void(0);) private static string[] ReadWriteHosts = System.Configuration.ConfigurationSettings.AppSettings["readWriteHosts"].Split(new char[] { ';' });

private static string[] ReadOnlyHosts = System.Configuration.ConfigurationSettings.AppSettings["readOnlyHosts"].Split(new char[] { ';' });

#region -- 连接信息 --

public static PooledRedisClientManager prcm = CreateManager(ReadWriteHosts, ReadOnlyHosts);

private static PooledRedisClientManager CreateManager(string[] readWriteHosts, string[] readOnlyHosts)

{

// 支持读写分离，均衡负载

return new PooledRedisClientManager(readWriteHosts, readOnlyHosts, new RedisClientManagerConfig

{

MaxWritePoolSize = 5, // “写”链接池链接数

MaxReadPoolSize = 5, // “读”链接池链接数

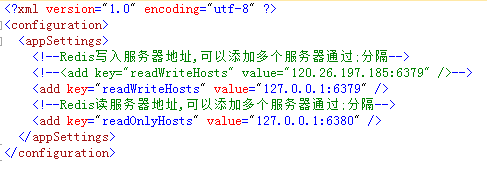
AutoStart = true,

});

}

[复制代码](javascript:void(0);)

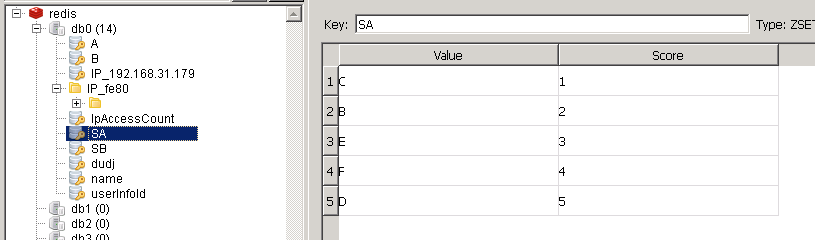
　 配置文件



**总结**

　1.其实php,java等多种语言都能使用redis,在我接触的项目中见到使用redis做为消息队列和缓存组件，当然它的功能远不止于此。后面的文章将详细介绍redis的几个使用案例。

      2.可以使用[redis desktop manager](http://redisdesktop.com/download" \t "_blank)管理工具查看服务器缓存中的数据



      本篇文章用到的资源打包下载地址：[redis\_demo](http://files.cnblogs.com/files/yanweidie/ResidSessionDemo.rar" \t "_blank)

　　svn下载地址：<http://code.taobao.org/svn/ResidSessionDemo/>