https://www.cnblogs.com/mushroom/p/4217541.html

[c#实现redis客户端(一)](https://www.cnblogs.com/mushroom/p/4217541.html)

  最近项目使用中要改造redis客户端，看了下文档，总结分享一下。

**阅读目录：**

1. [协议规范](https://www.cnblogs.com/mushroom/p/4217541.html#one)
2. [基础通信](https://www.cnblogs.com/mushroom/p/4217541.html#two)
3. [状态命令](https://www.cnblogs.com/mushroom/p/4217541.html#three)
4. [set、get命令](https://www.cnblogs.com/mushroom/p/4217541.html#four)
5. [管道、事务](https://www.cnblogs.com/mushroom/p/4217541.html#five)
6. [总结](https://www.cnblogs.com/mushroom/p/4217541.html#six)

**协议规范**

redis允许客户端以TCP方式连接，默认6379端口。传输数据都以\r\n结尾。

**请求格式**

\*<number of arguments>\r\n$<number of bytes of argument 1>\r\n*<argument data>\r\n*

例：\*1\r\n$4\r\nINFO\r\n

**响应格式**

1：简单字符串，非二进制安全字符串，一般是状态回复。  +开头，例：+OK\r\n

2: 错误信息。　　　　　　　　　　-开头， 例：-ERR unknown command 'mush'\r\n

3: 整型数字。                            :开头， 例：:1\r\n

4：大块回复值，最大512M。           $开头+数据长度。 例：$4\r\mush\r\n

5：多条回复。                           \*开头， 例：\*2\r\n$3\r\nfoo\r\n$3\r\nbar\r\n

**基础通信**

定义配置类：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | public class Configuration      {          public string Host { get; set; }          public int Port { get; set; }          /// <summary>          /// Socket 是否正在使用 Nagle 算法。          /// </summary>          public bool NoDelaySocket { get; set; }            public Configuration()          {              Host = "localhost";              Port = 6379;              NoDelaySocket = false;          }      } |

实现socket连接：

[复制代码](javascript:void(0);)

public class RedisBaseClient

{

//配置文件

private Configuration configuration;

//通信socket

private Socket socket;

//接收字节数组

private byte[] ReceiveBuffer = new byte[100000];

public RedisBaseClient(Configuration config)

{

configuration = config;

}

public RedisBaseClient()

: this(new Configuration())

{

}

public void Connect()

{

if (socket != null && socket.Connected)

return;

socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)

{

NoDelay = configuration.NoDelaySocket

};

socket.Connect(configuration.Host, configuration.Port);

if (socket.Connected)

return;

Close();

}

/// <summary>

/// 关闭client

/// </summary>

public void Close()

{

socket.Disconnect(false);

socket.Close();

}

}

[复制代码](javascript:void(0);)

调用：

RedisBaseClient redis = new RedisBaseClient();

redis.Connect();

服务端成功响应：

https://images0.cnblogs.com/blog/307762/201501/120031479065665.png

**状态命令**

定义Redis命令枚举：

[复制代码](javascript:void(0);)

public enum RedisCommand

{

GET, //获取一个key的值

INFO, //Redis信息。

SET, //添加一个值

EXPIRE, //设置过期时间

MULTI, //标记一个事务块开始

EXEC, //执行所有 MULTI 之后发的命令

}

[复制代码](javascript:void(0);)

发送命令构建：

[复制代码](javascript:void(0);)

public string SendCommand(RedisCommand command, params string[] args)

{

//请求头部格式， \*<number of arguments>\r\n

const string headstr = "\*{0}\r\n";

//参数信息 $<number of bytes of argument N>\r\n<argument data>\r\n

const string bulkstr = "${0}\r\n{1}\r\n";

var sb = new StringBuilder();

sb.AppendFormat(headstr, args.Length + 1);

var cmd = command.ToString();

sb.AppendFormat(bulkstr, cmd.Length, cmd);

foreach (var arg in args)

{

sb.AppendFormat(bulkstr, arg.Length, arg);

}

byte[] c = Encoding.UTF8.GetBytes(sb.ToString());

try

{

Connect();

socket.Send(c);

socket.Receive(ReceiveBuffer);

Close();

return ReadData();

}

catch (SocketException e)

{

Close();

}

return null;

}

private string ReadData()

{

var data = Encoding.UTF8.GetString(ReceiveBuffer);

char c = data[0];

//错误消息检查。

if (c == '-') //异常处理。

throw new Exception(data);

//状态回复。

if (c == '+')

return data;

return data;

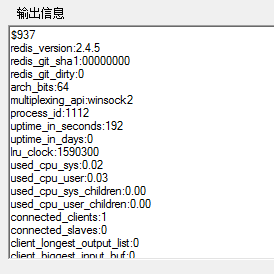
}

[复制代码](javascript:void(0);)

 调用：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | private void button1\_Click(object sender, EventArgs e)         {             RedisBaseClient redis = new RedisBaseClient();             var result = redis.SendCommand(RedisCommand.INFO);             richTextBox1.Text = result;         } |

输出响应，其$937是数据包的长度。



**set、get命令**

调用：

[复制代码](javascript:void(0);)

private void button2\_Click(object sender, EventArgs e)

{

RedisBaseClient redis = new RedisBaseClient();

var result = redis.SendCommand(RedisCommand.SET, "msg", "testvalue");

richTextBox1.Text = result.ToString();

}

private void button3\_Click(object sender, EventArgs e)

{

RedisBaseClient redis = new RedisBaseClient();

var result = redis.SendCommand(RedisCommand.GET, "msg");

richTextBox1.Text = result.ToString();

}

[复制代码](javascript:void(0);)

输出

https://images0.cnblogs.com/blog/307762/201501/120053525782635.png

**管道、事务**

 二者都是走的MULTI，EXEC命令，原子操作。管道就是发送命令(无需等上次命令回复)，进入命令队列，然后多条命令一次执行，并返回客户端结果。

 平常使用ServiceStack.Redis客户端都直接set了，其实是set、expire 2个命令。 简单实现如下：

[复制代码](javascript:void(0);)

public void CreatePipeline()

{

SendCommand(RedisCommand.MULTI, new string[] {}, true);

}

public string EnqueueCommand(RedisCommand command, params string[] args)

{

return SendCommand(command, args, true);

}

public string FlushPipeline()

{

var result = SendCommand(RedisCommand.EXEC, new string[] {}, true);

Close();

return result;

}

public string SendCommand(RedisCommand command, string[] args, bool isPipeline=false)

{

//请求头部格式， \*<number of arguments>\r\n

const string headstr = "\*{0}\r\n";

//参数信息 $<number of bytes of argument N>\r\n<argument data>\r\n

const string bulkstr = "${0}\r\n{1}\r\n";

var sb = new StringBuilder();

sb.AppendFormat(headstr, args.Length + 1);

var cmd = command.ToString();

sb.AppendFormat(bulkstr, cmd.Length, cmd);

foreach (var arg in args)

{

sb.AppendFormat(bulkstr, arg.Length, arg);

}

byte[] c = Encoding.UTF8.GetBytes(sb.ToString());

try

{

Connect();

socket.Send(c);

socket.Receive(ReceiveBuffer);

if (!isPipeline)

{

Close();

}

return ReadData();

}

catch (SocketException e)

{

Close();

}

return null;

}

public string SetByPipeline(string key, string value, int second)

{

this.CreatePipeline();

this.EnqueueCommand(RedisCommand.SET, key, value);

this.EnqueueCommand(RedisCommand.EXPIRE, key, second.ToString());

return this.FlushPipeline();

}

[复制代码](javascript:void(0);)

 调用：

private void button4\_Click(object sender, EventArgs e)

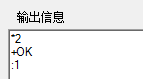
{

RedisBaseClient redis = new RedisBaseClient();

richTextBox1.Text = redis.SetByPipeline("cnblogs", "mushroom", 1000);

}

输出：



\*2 表示2条回复。

+2 表示命令执行OK。

:1  表示命令执行的结果

**总结**

本文只是简单的实现，有兴趣的同学，可以继续下去。

客户端实现这块，Socket连接池管理相较复杂些。

参考资源：

http://redis.io/topics/protocol

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