GitHub地址（不可用）：https://github.com/StackExchange/StackExchange.Redis

StackExchange.Redis

For all documentation, see here.

Build Status

build failing

Package Status

MyGet Pre-release feed: https://www.myget.org/gallery/stackoverflow

Package	NuGet Stable	NuGet Pre-release	MyGet
StackExchange.Redis	nuget v2.0.519	nuget v2.0.519	stackoverflow v2.0.519
NRediSearch	nuget v1.0.0	nuget v2.0.0-alpha.169	

StackExchange.Redis 实现了对 redis cluster（集群）的支持

使用集群

Redis 集群现阶段的一个问题是客户端实现很少。

以下是一些我知道的实现：

- redis-rb-cluster 是我（@antirez）编写的 Ruby 实现， 用于作为其他实现的参考。 该实现是对 redis-rb 的一个简单包装， 高效地实现了与集群进行通讯所需的最少语义（semantic）.
- redis-py-cluster 看上去是 redis-rb-cluster 的一个 Python 版本， 这个项目有一段时间没有更新了（最后一次提交是在六个月之前）， 不过可以将这个项目用作学习集群的起点。
- 流行的 Predis 曾经对早期的 Redis 集群有过一定的支持， 但我不确定它对集群的支持是否完整， 也不清楚它是否和最新版本的 Redis 集群兼容 （因为新版的 Redis 集群将槽的数量从 4k 改为 16k 了）.
- 使用最多的时java客户端, Jedis 最近添加了对集群的支持, 详细请查看项目README中*Jedis Cluster*部分.
- StackExchange.Redis 提供对 C# 的支持(并且包括大部分 .NET 下面的语言，比如： VB, F#等等)
- thunk-redis 提供对 Node.js 和 io.js的支持。
- Redis unstable 分支中的 `redis-cli` 程序实现了非常基本的集群支持， 可以使用命令 redis-cli -c 来启动。

国内封装的Helper GitHub地址（实际可用）：https://gitee.com/wdj8401/Wdj.Redis.Helper.git

该项目已经产品化为：eWorld.Redis.Helper，源码可以找孔老

以下为eWorld.Redis.Helper实际的项目配置：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1"/>
  </startup>
  <appSettings>
    <!--Redis保存的Key前缀，会自动添加到指定的Key名称前-->
    <!--各个子系统网站使用不同的前缀-->
    <add key="RedisSysCustomKey" value="eWorldCloud_CRM"/>
    <!--当前连接的Redis中的DataBase索引，默认0-64，可以在service.conf配置，最高64-->
    <add key="RedisDataBaseIndex" value="0"/>
    <!--当前连接的Redis中连接字符串，格式为：127.0.0.1:6379,allowadmin=true,passowrd=pwd-->
    <add key="RedisHostConnection"
value="192.168.2.81:7001,192.168.2.82:7001,192.168.2.83:7001,192.168.2.81:7002,192.168.2.82:7002,192.168.2.83:7002,allowadm
  </appSettings>
</configuration>
```

StackExchange.Redis中文使用文档--配置

配置

因为有很多不同配置 redis 的方式，StackExchange.Redis 提供了一个丰富的配置模型，当调用 `Connect` （或 `ConnectAsync` ）时调用它。

```
var conn = ConnectionMultiplexer.Connect(configuration);
```

这里的 `configuration` 可以是下面的任意一个：

- 一个 ConfigurationOptions 实例
- 一个代表配置的 string

后者是 *基本上* 是前者的标记化形式。

基本配置字符串

最简单的 配置示例只需要一个主机名：

```
var conn = ConnectionMultiplexer.Connect("localhost");
```

这将使用默认的redis端口（6379）连接到本地计算机上的单个服务器。

附加选项只是简单地附加（逗号分隔）。 端口通常用冒号（:）表示。 配置*选项*在名称后面包含一个=。 例如：

```
var conn = ConnectionMultiplexer.Connect("redis0:6380,redis1:6380,allowAdmin=true");
```

下面显示了 string 和 ConfigurationOptions 表示之间的映射概述，但您可以轻松地在它们之间切换：

```
ConfigurationOptions options = ConfigurationOptions.Parse(configString);
```

或者：

```
string configString = options.ToString();
```

常见的用法是将 *基础配置* 细节存储在一个字符串中，然后在运行时应用特定的详细信息：

```
string configString = GetRedisConfiguration();
var options = ConfigurationOptions.Parse(configString);
options.ClientName = GetAppName(); // only known at runtime
options.AllowAdmin = true;
conn = ConnectionMultiplexer.Connect(options);
```

带密码的 Microsoft Azure Redis 示例

```
var conn = ConnectionMultiplexer.Connect("contoso5.redis.cache.windows.net,ssl=true,password=...");
```

配置选项

ConfigurationOptions对象具有许多的属性，所有这些都在智能提示中都有。

一些更常用的选项包括：

配置字符串	ConfigurationOptions	含义
abortConnect={bool}	AbortOnConnectFail	如果为true，Connect 没有服务器可用时将不会创建连接
allowAdmin={bool}	AllowAdmin	启用被认为具有风险的一系列命令
channelPrefix={string}	ChannelPrefix	所有发布/订阅操作的可选频道前缀
connectRetry={int}	ConnectRetry	在初始 Connect 期间重复连接尝试的次数
connectTimeout={int}	ConnectTimeout	连接操作的超时时间（ms）
configChannel={string}	ConfigurationChannel	用于传达配置更改的广播通道名称
defaultDatabase={int}	DefaultDatabase	默认数据库索引，从 0 到 databases - 1（0 到 Databases.Count -1）
keepAlive={int}	KeepAlive	发送消息以帮助保持套接字活动的时间（秒）
name={string}	ClientName	标识 redis 中的连接
password={string}	Password	redis 服务器的密码
proxy={proxy type}	Proxy	正在使用的代理类型（如果有）；例如 “twemproxy”
resolveDns={bool}	ResolveDns	指定DNS解析应该是显式和热切，而不是隐式
serviceName={string}	ServiceName	目前尚未实施（预期与sentinel一起使用）
ssl={bool}	Ssl	指定应使用SSL加密
sslHost={string}	SslHost	在服务器证书上强制执行特定的SSL主机标识
syncTimeout={int}	SyncTimeout	允许同步操作的时间（ms）
tiebreaker={string}	TieBreaker	用于在不明确的主场景中选择服务器的键
version={string}	DefaultVersion	Redis版本级别（当服务器要使用的版本默认不可用时使用）

writeBuffer={int}	WriteBuffer	输出缓冲区的大小
ReconnectRetryPolicy={IReconnectRetryPolicy}	ReconnectRetryPolicy	重新连接重试策略

配置字符串中的令牌是逗号分隔的;任何没有=符号的都假定为redis服务器端点。

没有显式端口的端点将在未启用ssl的情况下使用6379，如果启用了ssl则使用6380。

以$开头的令牌被用来表示命令映射，例如：$ config = cfg。

自动和手动配置

在许多常见的情况下，StackExchange.Redis将自动配置很多设置，包括服务器类型和版本，连接超时和主/从关系。

有时，在redis服务器上禁用了这些命令。 在这种情况下，可以提供更多的信息：

```
ConfigurationOptions config = new ConfigurationOptions
{
    EndPoints =
    {
        { "redis0", 6379 },
        { "redis1", 6380 }
    },
    CommandMap = CommandMap.Create(new HashSet<string>
    { // EXCLUDE a few commands
        "INFO", "CONFIG", "CLUSTER",
        "PING", "ECHO", "CLIENT"
    }, available: false),
    KeepAlive = 180,
    DefaultVersion = new Version(2, 8, 8),
    Password = "changeme"
};
```

它相当于命令字符串：

```
redis0:6379,redis1:6380,keepAlive=180,version=2.8.8,$CLIENT=,$CLUSTER=,$CONFIG=,$ECHO=,$INFO=,$PING=
```

重命名命令

redis的一个很不寻常的功能是可以禁用或重命名或禁用并重命名单个命令。

根据前面的例子，这是通过CommandMap来实现的，但不是传递一个HashSet <string>到Create（） （表示可用或不可用的命令），而是传递一个Dictionary < string>。

字典中未提及的所有命令都默认已启用且未重命名。

“null”或空白值记录命令被禁用。 例如：

```
var commands = new Dictionary<string,string> {
        { "info", null }, // disabled
        { "select", "use" }, // renamed to SQL equivalent for some reason
};
var options = new ConfigurationOptions {
    // ...
    CommandMap = CommandMap.Create(commands),
    // ...
}
```

以上代码等同于（在连接字符串中）：

```
$INFO=,$SELECT=use
```

Twemproxy（3.0以前版本使用代理模式，不推荐使用）

Twemproxy 是允许使用多个 redis 实例就像它是一个单个服务器，具有内置分片和容错（很像 redis 集群，但单独实现）的工具。

Twemproxy可用的功能集减少。

为了避免手动配置，可以使用 Proxy 选项：

```
var options = new ConfigurationOptions
{
    EndPoints = { "my-server" },
    Proxy = Proxy.Twemproxy
};
```

Tiebreakers 和配置更改公告

通常 StackExchange.Redis 都会自动解析 主/从节点。然而，如果你不使用诸如 redis-sentinel 或 redis 集群之类的管理工具，有时你会有多个主节点（例如，在重置节点以进行维护时，它可能会作为主节点重新显示在网络上）。

为了帮助解决这个问题，StackExchange.Redis 可以使用 *tie-breaker* 的概念 - 这仅在检测到多个主节点时使用（不包括需要多个主节点的redis集群）。

为了与 BookSleeve 兼容，tiebreaker 使用默认的key为 `"__Booksleeve_TieBreak"` （总是在数据库0中）。

这用作粗略的投票机制，以帮助确定*首选* 主机，以便能够按正确的路由工作。

同样，当配置改变时（特别是主/从配置），连接的实例使得他们意识到新的情况（在可用的地方通过 `INFO`, `CONFIG` 等进行通知 ）是很重要的。

StackExchange.Redis 通过自动订阅可以在其上发送这样的通知的发布/订阅通道来实现。

由于类似的原因，这默认为`"__Booksleeve_MasterChanged"`。

这两个选项都可以通过 `.ConfigurationChannel` 和 `.TieBreaker` 配置属性来定制或禁用（设置为`""`）。

These settings are also used by the `IServer.MakeMaster()` method, which can set the tie-breaker in the database and broadcast the configuration change message.

The configuration message can also be used separately to master/slave changes simply to request all nodes to refresh their configurations, via the `ConnectionMultiplexer.PublishReconfigure` method.

这些设置也由 `IServer.MakeMaster()` 方法使用，它可以设置数据库中的 tie-breaker 并广播配置更改消息。

配置消息也可以单独用于主/从变化，只需通过调用 `ConnectionMultiplexer.PublishReconfigure` 方法请求所有节点刷新其配置。

重新连接重试策略

当连接由于任何原因丢失时，StackExchange.Redis会自动尝试在后台重新连接。

它将继续重试，直到连接恢复。 它将使用 ReconnectRetryPolicy 来决定在重试之间应该等待多长时间。

ReconnectRetryPolicy可以是线性的（默认），指数的或者是一个自定义的重试策略。

举个例子:

```
config.ReconnectRetryPolicy = new ExponentialRetry(5000); // defaults maxDeltaBackoff to 10000 ms
//retry#    retry to re-connect after time in milliseconds
//1         a random value between 5000 and 5500
//2         a random value between 5000 and 6050
//3         a random value between 5000 and 6655
//4         a random value between 5000 and 8053
//5         a random value between 5000 and 10000, since maxDeltaBackoff was 10000 ms
//6         a random value between 5000 and 10000

config.ReconnectRetryPolicy = new LinearRetry(5000);
//retry#    retry to re-connect after time in milliseconds
//1         5000
//2         5000
//3         5000
//4         5000
//5         5000
//6         5000
```

查看原文