**C#客户端Redis服务器的分布式缓存**

在这篇文章中，我想介绍我知道的一种最紧凑的安装和配置Redis服务器的方式。另外，我想简短地概述一下在.NET / C#客户端下Redis hash（哈希类型）和list（链表）的使用。

作者：小峰来源：码农网|2015-08-17 09:48

[收藏](javascript:favorBox('open');)

[分享](javascript:;)

**介绍**

在这篇文章中，我想介绍我知道的一种最紧凑的安装和配置Redis服务器的方式。另外，我想简短地概述一下在.NET / C#客户端下Redis hash（哈希类型）和list（链表）的使用。

在这篇文章主要讲到：

* 安装Redis服务器（[附完整的应用程序文件设置](http://www.codeproject.com/KB/web-cache/636730/Redis.zip)）
* Redis服务器保护（配置身份验证）
* 配置服务器复制
* 从C#应用程序访问缓存
* 使用Redis ASP.NET会话状态
* Redis 集合（Set）、列表（List）和事务处理用法示例
* 说明附加的源（Redis Funq LoC MVC项目：举例）
* 缓存的优化思路

**背景**

Redis是最快也是功能最丰富的内存Key-Value数据存储系统之一。

**缺点**

* 没有本地数据缓存（如在Azure缓存同步本地数据缓存）
* 没有完全集群化的支持（不过，可能今年年底会实现）

**优点**

* 易于配置
* 使用简单
* 高性能
* 支持不同的数据类型（如hash（哈希类型）、list（链表）、set（集合）、sorted set（有序集））
* ASP.NET会话集成
* Web UI用于浏览缓存内容

下面我将简单说明如何在服务器上安装和配置Redis，并用C#使用它。

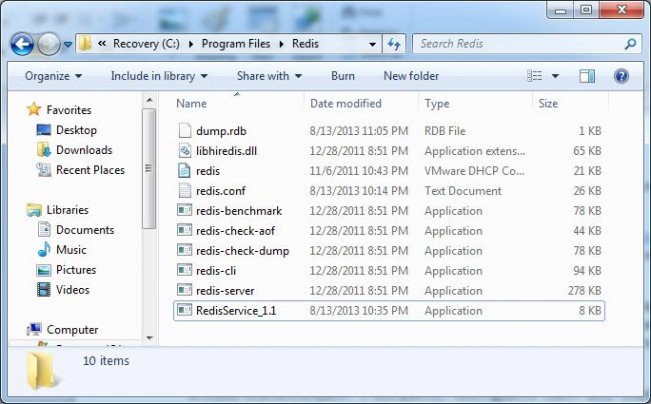
**Redis的安装**

从<https://github.com/dmajkic/redis/downloads>（win32 win64直接链接）下载二进制文件，解包档案到应用程序目录（如C：\Program Files\Redis）

下载从<https://github.com/kcherenkov/redis-windows-service/downloads>编译的Redis服务，然后复制到程序文件夹（如C：\Program Files\Redis）。如果配置文件丢失，也可以下载复制到应用程序目录。有效的Redis配置文件的范例在[https://raw.github.com/antirez/redis/2.6/redis.conf](http://raw.github.com/antirez/redis/2.6/redis.conf)。

Redis应用程序的完整文件也可以从压缩文件（x64）得到。

当你拥有了全套的应用程序文件（如下图所示），



**导航到应用程序目录，然后运行以下命令：**

sc create %name% binpath= "\"%binpath%\" %configpath%" start= "auto" DisplayName= "Redis"

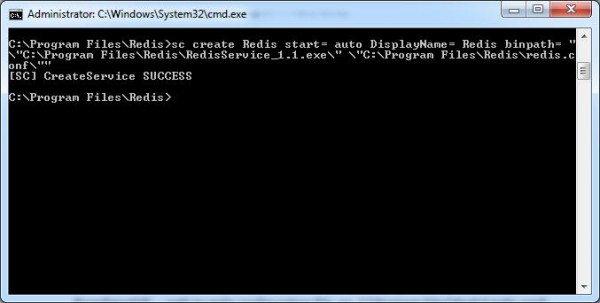
**其中：**

* %name%——服务实例的名称，例如：redis-instance；
* %binpath%——到项目exe文件的路径，例如：C：\Program Files\Redis\RedisService\_1.1.exe；
* %configpath%——到Redis配置文件的路径，例如：C：\Program Files\Redis\redis.conf；

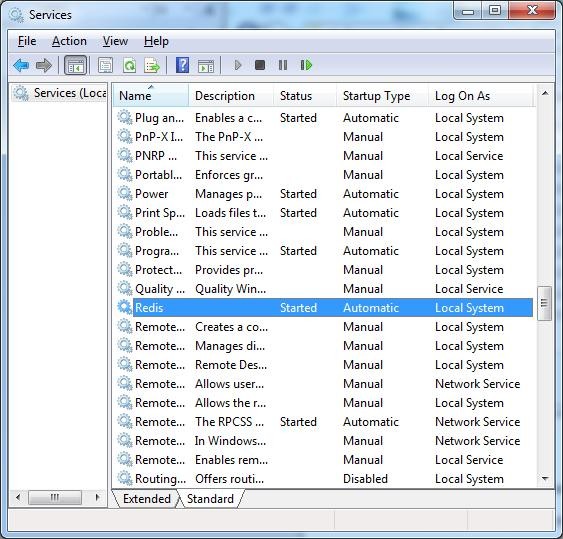
**举例：**

sc create Redis start= auto DisplayName= Redis binpath= "\"C:\Program Files\Redis\RedisService\_1.1.exe\  
" \"C:\Program Files\Redis\redis.conf\""

**即应该是这样的：**



请确保有足够的权限启动该服务。安装完毕后，请检查该服务是否创建成功，当前是否正在运行：



或者，你可以使用安装程序（我没试过）：<https://github.com/rgl/redis/downloads>。

**Redis服务器保护：密码，IP过滤**

保护Redis服务器的主要方式是使用Windows防火墙或活跃的网络连接属性设置IP过滤。此外，还可以使用Redis密码设置额外保护。这需要用下面的方式更新Redis配置文件（redis.conf）：

**首先，找到这行：**

# requirepass foobared

删除开头的#符号，用新密码替换foobared：

requirepass foobared

然后，重新启动Redis Windows服务！

**当具体使用客户端的时候，使用带密码的构造函数：**

RedisClient client = new RedisClient(serverHost, port, redisPassword);

**Redis服务器复制（主—从配置）**

Redis支持主从同步，即，每次主服务器修改，从服务器得到通知，并自动同步。大多复制用于读取（但不能写）扩展和数据冗余和服务器故障转移。设 置两个Redis实例（在相同或不同服务器上的两个服务），然后配置其中之一作为从站。为了让Redis服务器实例是另一台服务器的从属，可以这样更改配 置文件：

**找到以下代码：**

# slaveof <masterip> <masterport>

**替换为：**

slaveof 192.168.1.1 6379

（可以自定义指定主服务器的真实IP和端口）。如果主服务器配置为需要密码（验证），可以如下所示改变redis.conf，找到这一行代码：

# masterauth <master-password>

删除开头的#符号，用主服务器的密码替换<master-password>，即：

masterauth mastpassword

现在这个Redis实例可以被用来作为主服务器的只读同步副本。

**用C#代码使用Redis缓存**

用C#代码使用Redis运行Manage NuGet包插件，找到ServiceStack.Redis包，并进行安装。



直接从实例化客户端使用Set/Get方法示例：

1. string host = "localhost";
2. string elementKey = "testKeyRedis";
4. using (RedisClient redisClient = **new** RedisClient(host))
5. {
6. **if** (redisClient.Get<string>(elementKey) == **null**)
7. {
8. // adding delay to see the difference
9. Thread.Sleep(5000);
10. // save value in cache
11. redisClient.Set(elementKey, "some cached value");
12. }
13. // get value from the cache by key
14. message = "Item value is: " + redisClient.Get<string>("some cached value");
15. }

类型化实体集更有意思和更实用，这是因为它们操作的是确切类型的对象。在下面的代码示例中，有两个类分别定义为Phone和Person——phone的主人。每个phone实例引用它的主人。下面的代码演示我们如何通过标准添加、删除和发现缓存项：

1. **public** **class** Phone
2. {
3. **public** **int** Id { get; set; }
4. **public** string Model { get; set; }
5. **public** string Manufacturer { get; set; }
6. **public** Person Owner { get; set; }
7. }
9. **public** **class** Person
10. {
11. **public** **int** Id { get; set; }
12. **public** string Name { get; set; }
13. **public** string Surname { get; set; }
14. **public** **int** Age { get; set; }
15. **public** string Profession { get; set; }
16. }
18. using (RedisClient redisClient = **new** RedisClient(host))
19. {
20. IRedisTypedClient<phone> phones = redisClient.As<phone>();
21. Phone phoneFive = phones.GetValue("5");
22. **if** (phoneFive == **null**)
23. {
24. // make a small delay
25. Thread.Sleep(5000);
26. // creating a new Phone entry
27. phoneFive = **new** Phone
28. {
29. Id = 5,
30. Manufacturer = "Motorolla",
31. Model = "xxxxx",
32. Owner = **new** Person
33. {
34. Id = 1,
35. Age = 90,
36. Name = "OldOne",
37. Profession = "sportsmen",
38. Surname = "OldManSurname"
39. }
40. };
41. // adding Entry to the typed entity set
42. phones.SetEntry(phoneFive.Id.ToString(), phoneFive);
43. }
44. message = "Phone model is " + phoneFive.Manufacturer;
45. message += "Phone Owner Name is: " + phoneFive.Owner.Name;
46. }

在上面的例子中，我们实例化了输入端IRedisTypedClient，它与缓存对象的特定类型——Phone类型一起工作。

Redis ASP.NET会话状态

要用Redis提供商配置ASP.NET会话状态，添加新文件到你的Web项目，命名为RedisSessionStateProvider.cs，可以从[https://github.com/chadman/redis-service-provider/raw/master/RedisProvider/SessionProvider/RedisSessionProvider.cs](https://github.com/chadman/redis-service-provider/raw/master/RedisProvider/SessionProvider/RedisSessionProvider.cs%E5%A4%8D%E5%88%B6%E4%BB%A3%E7%A0%81)复制代码，然后添加或更改配置文件中的以下部分（sessionState标签已经内置于system.web标签），或者你也可以下载附加来源和复制代码。

1. <sessionstate timeout="1" mode="Custom"
2. customprovider="RedisSessionStateProvider" cookieless="false">
3. <providers>
4. <add name="RedisSessionStateProvider" writeexceptionstoeventlog="false"
5. type="RedisProvider.SessionProvider.CustomServiceProvider"
6. server="localhost" port="6379" password="pasword">
7. </add> </providers>
8. </sessionstate>

注意，此密码是可以选择的，看服务器是否需要认证。它必须被真实的值替换或删除，如果Redis服务器不需要身份验证，那么服务器属性和端口得由具体的数值代替（默认端口为6379）。然后在项目中，你才可以使用会话状态：

1. // in the Global.asax
2. **public** **class** MvcApplication1 : System.Web.HttpApplication
3. {
4. **protected** **void** Application\_Start()
5. {
6. //....
7. }
9. **protected** **void** Session\_Start()
10. {
11. Session["testRedisSession"] = "Message from the redis ression";
12. }
13. }
15. 在Home controller（主控制器）：
17. **public** **class** HomeController : Controller
18. {
19. **public** ActionResult Index()
20. {
21. //...
22. ViewBag.Message = Session["testRedisSession"];
23. **return** View();
24. }
25. //...
26. }

结果：



ASP.NET输出缓存提供者，并且Redis可以用类似的方式进行配置。

**Redis Set（集合）和List（列表）**

主要要注意的是，Redis列表实现IList<T>，而Redis集合实现ICollection<T>。下面来说说如何使用它们。

当需要区分相同类型的不同分类对象时，使用列表。例如，我们有“mostSelling（热销手机）”和“oldCollection（回收手机）”两个列表：

1. string host = "localhost";
2. using (var redisClient = **new** RedisClient(host))
3. {
4. //Create a 'strongly-typed' API that makes all Redis Value operations to apply against Phones
5. IRedisTypedClient<phone> redis = redisClient.As<phone>();
7. IRedisList<phone> mostSelling = redis.Lists["urn:phones:mostselling"];
8. IRedisList<phone> oldCollection = redis.Lists["urn:phones:oldcollection"];
10. Person phonesOwner = **new** Person
11. {
12. Id = 7,
13. Age = 90,
14. Name = "OldOne",
15. Profession = "sportsmen",
16. Surname = "OldManSurname"
17. };
19. // adding new items to the list
20. mostSelling.Add(**new** Phone
21. {
22. Id = 5,
23. Manufacturer = "Sony",
24. Model = "768564564566",
25. Owner = phonesOwner
26. });
28. oldCollection.Add(**new** Phone
29. {
30. Id = 8,
31. Manufacturer = "Motorolla",
32. Model = "324557546754",
33. Owner = phonesOwner
34. });
36. var upgradedPhone  = **new** Phone
37. {
38. Id = 3,
39. Manufacturer = "LG",
40. Model = "634563456",
41. Owner = phonesOwner
42. };
44. mostSelling.Add(upgradedPhone);
46. // remove item from the list
47. oldCollection.Remove(upgradedPhone);
49. // find objects in the cache
50. IEnumerable<phone> LGPhones = mostSelling.Where(ph => ph.Manufacturer == "LG");
52. // find specific
53. Phone singleElement = mostSelling.FirstOrDefault(ph => ph.Id == 8);
55. //reset sequence and delete all lists
56. redis.SetSequence(0);
57. redisClient.Remove("urn:phones:mostselling");
58. redisClient.Remove("urn:phones:oldcollection");
59. }

当需要存储相关的数据集和收集统计信息，例如answer -> queustion给答案或问题投票时，Redis集合就非常好使。假设我们有很多的问题（queustion）和答案（answer ），需要将它们存储在缓存中。使用Redis，我们可以这么做：

1. /// <summary>
2. /// Gets or sets the Redis Manager. The built-in IoC used with ServiceStack autowires this property.
3. /// </summary>
4. IRedisClientsManager RedisManager { get; set; }
5. /// <summary>
6. /// Delete question by performing compensating actions to
7. /// StoreQuestion() to keep the datastore in a consistent state
8. /// </summary>
9. /// <param name="questionId">
10. **public** **void** DeleteQuestion(**long** questionId)
11. {
12. using (var redis = RedisManager.GetClient())
13. {
14. var redisQuestions = redis.As<question>();
16. var question = redisQuestions.GetById(questionId);
17. **if** (question == **null**) **return**;
19. //decrement score in tags list
20. question.Tags.ForEach(tag => redis.IncrementItemInSortedSet("urn:tags", tag, -1));
22. //remove all related answers
23. redisQuestions.DeleteRelatedEntities<answer>(questionId);
25. //remove this question from user index
26. redis.RemoveItemFromSet("urn:user>q:" + question.UserId, questionId.ToString());
28. //remove tag => questions index for each tag
29. question.Tags.ForEach("urn:tags>q:" + tag.ToLower(), questionId.ToString()));
31. redisQuestions.DeleteById(questionId);
32. }
33. }
35. **public** **void** StoreQuestion(Question question)
36. {
37. using (var redis = RedisManager.GetClient())
38. {
39. var redisQuestions = redis.As<question>();
41. **if** (question.Tags == **null**) question.Tags = **new** List<string>();
42. **if** (question.Id == **default**(**long**))
43. {
44. question.Id = redisQuestions.GetNextSequence();
45. question.CreatedDate = DateTime.UtcNow;
47. //Increment the popularity for each new question tag
48. question.Tags.ForEach(tag => redis.IncrementItemInSortedSet("urn:tags", tag, 1));
49. }
51. redisQuestions.Store(question);
52. redisQuestions.AddToRecentsList(question);
53. redis.AddItemToSet("urn:user>q:" + question.UserId, question.Id.ToString());
55. //Usage of tags - Populate tag => questions index for each tag
56. question.Tags.ForEach(tag => redis.AddItemToSet
57. ("urn:tags>q:" + tag.ToLower(), question.Id.ToString()));
58. }
59. }
61. /// <summary>
62. /// Delete Answer by performing compensating actions to
63. /// StoreAnswer() to keep the datastore in a consistent state
64. /// </summary>
65. /// <param name="questionId">
66. /// <param name="answerId">
67. **public** **void** DeleteAnswer(**long** questionId, **long** answerId)
68. {
69. using (var redis = RedisManager.GetClient())
70. {
71. var answer = redis.As<question>().GetRelatedEntities<answer>
72. (questionId).FirstOrDefault(x => x.Id == answerId);
73. **if** (answer == **null**) **return**;
75. redis.As<question>().DeleteRelatedEntity<answer>(questionId, answerId);
77. //remove user => answer index
78. redis.RemoveItemFromSet("urn:user>a:" + answer.UserId, answerId.ToString());
79. }
80. }
82. **public** **void** StoreAnswer(Answer answer)
83. {
84. using (var redis = RedisManager.GetClient())
85. {
86. **if** (answer.Id == **default**(**long**))
87. {
88. answer.Id = redis.As<answer>().GetNextSequence();
89. answer.CreatedDate = DateTime.UtcNow;
90. }
92. //Store as a 'Related Answer' to the parent Question
93. redis.As<question>().StoreRelatedEntities(answer.QuestionId, answer);
94. //Populate user => answer index
95. redis.AddItemToSet("urn:user>a:" + answer.UserId, answer.Id.ToString());
96. }
97. }
99. **public** List<answer> GetAnswersForQuestion(**long** questionId)
100. {
101. using (var redis = RedisManager.GetClient())
102. {
103. **return** redis.As<question>().GetRelatedEntities<answer>(questionId);
104. }
105. }
107. **public** **void** VoteQuestionUp(**long** userId, **long** questionId)
108. {
109. //Populate Question => User and User => Question set indexes in a single transaction
110. RedisManager.ExecTrans(trans =>
111. {
112. //Register upvote against question and remove any downvotes if any
113. trans.QueueCommand(redis =>
114. redis.AddItemToSet("urn:q>user+:" + questionId, userId.ToString()));
115. trans.QueueCommand(redis =>
116. redis.RemoveItemFromSet("urn:q>user-:" + questionId, userId.ToString()));
118. //Register upvote against user and remove any downvotes if any
119. trans.QueueCommand(redis =>
120. redis.AddItemToSet("urn:user>q+:" + userId, questionId.ToString()));
121. trans.QueueCommand(redis =>
122. redis.RemoveItemFromSet("urn:user>q-:" + userId, questionId.ToString()));
123. });
124. }
126. **public** **void** VoteQuestionDown(**long** userId, **long** questionId)
127. {
128. //Populate Question => User and User => Question set indexes in a single transaction
129. RedisManager.ExecTrans(trans =>
130. {
131. //Register downvote against question and remove any upvotes if any
132. trans.QueueCommand(redis =>
133. redis.AddItemToSet("urn:q>user-:" + questionId, userId.ToString()));
134. trans.QueueCommand(redis =>
135. redis.RemoveItemFromSet("urn:q>user+:" + questionId, userId.ToString()));
137. //Register downvote against user and remove any upvotes if any
138. trans.QueueCommand(redis =>
139. redis.AddItemToSet"urn:user>q-:" + userId, questionId.ToString()));
140. trans.QueueCommand(redis =>
141. redis.RemoveItemFromSet("urn:user>q+:" + userId, questionId.ToString()));
142. });
143. }
145. **public** **void** VoteAnswerUp(**long** userId, **long** answerId)
146. {
147. //Populate Question => User and User => Question set indexes in a single transaction
148. RedisManager.ExecTrans(trans =>
149. {
150. //Register upvote against answer and remove any downvotes if any
151. trans.QueueCommand(redis =>
152. redis.AddItemToSet("urn:a>user+:" + answerId, userId.ToString()));
153. trans.QueueCommand(redis =>
154. redis.RemoveItemFromSet("urn:a>user-:" + answerId, userId.ToString()));
156. //Register upvote against user and remove any downvotes if any
157. trans.QueueCommand(redis =>
158. redis.AddItemToSet("urn:user>a+:" + userId, answerId.ToString()));
159. trans.QueueCommand(redis =>
160. redis.RemoveItemFromSet("urn:user>a-:" + userId, answerId.ToString()));
161. });
162. }
164. **public** **void** VoteAnswerDown(**long** userId, **long** answerId)
165. {
166. //Populate Question => User and User => Question set indexes in a single transaction
167. RedisManager.ExecTrans(trans =>
168. {
169. //Register downvote against answer and remove any upvotes if any
170. trans.QueueCommand(redis =>
171. redis.AddItemToSet("urn:a>user-:" + answerId, userId.ToString()));
172. trans.QueueCommand(redis =>
173. redis.RemoveItemFromSet("urn:a>user+:" + answerId, userId.ToString()));
175. //Register downvote against user and remove any upvotes if any
176. trans.QueueCommand(redis =>
177. redis.AddItemToSet("urn:user>a-:" + userId, answerId.ToString()));
178. trans.QueueCommand(redis =>
179. redis.RemoveItemFromSet("urn:user>a+:" + userId, answerId.ToString()));
180. });
181. }
183. **public** QuestionResult GetQuestion(**long** questionId)
184. {
185. var question = RedisManager.ExecAs<question>
186. (redisQuestions => redisQuestions.GetById(questionId));
187. **if** (question == **null**) **return** **null**;
189. var result = ToQuestionResults(**new**[] { question })[0];
190. var answers = GetAnswersForQuestion(questionId);
191. var uniqueUserIds = answers.ConvertAll(x => x.UserId).ToHashSet();
192. var usersMap = GetUsersByIds(uniqueUserIds).ToDictionary(x => x.Id);
194. result.Answers = answers.ConvertAll(answer =>
195. **new** AnswerResult { Answer = answer, User = usersMap[answer.UserId] });
197. **return** result;
198. }
200. **public** List<user> GetUsersByIds(IEnumerable<**long**> userIds)
201. {
202. **return** RedisManager.ExecAs<user>(redisUsers => redisUsers.GetByIds(userIds)).ToList();
203. }
205. **public** QuestionStat GetQuestionStats(**long** questionId)
206. {
207. using (var redis = RedisManager.GetReadOnlyClient())
208. {
209. var result = **new** QuestionStat
210. {
211. VotesUpCount = redis.GetSetCount("urn:q>user+:" +questionId),
212. VotesDownCount = redis.GetSetCount("urn:q>user-:" + questionId)
213. };
214. result.VotesTotal = result.VotesUpCount - result.VotesDownCount;
215. **return** result;
216. }
217. }
219. **public** List<tag> GetTagsByPopularity(**int** skip, **int** take)
220. {
221. using (var redis = RedisManager.GetReadOnlyClient())
222. {
223. var tagEntries = redis.GetRangeWithScoresFromSortedSetDesc("urn:tags", skip, take);
224. var tags = tagEntries.ConvertAll(kvp => **new** Tag { Name = kvp.Key, Score = (**int**)kvp.Value });
225. **return** tags;
226. }
227. }
229. **public** SiteStats GetSiteStats()
230. {
231. using (var redis = RedisManager.GetClient())
232. {
233. **return** **new** SiteStats
234. {
235. QuestionsCount = redis.As<question>().TypeIdsSet.Count,
236. AnswersCount = redis.As<answer>().TypeIdsSet.Count,
237. TopTags = GetTagsByPopularity(0, 10)
238. };
239. }
240. }

附加资源说明

项目中引用的一些包在packages.config文件中配置。

Funq IoC的相关配置，以及注册类型和当前控制器目录，在Global.asax文件中配置。

基于IoC的缓存使用以及Global.asax可以打开以下URL：http://localhost:37447/Question/GetQuestions?tag=test 查看。

你可以将tag字段设置成test3，test1，test2等。

Redis缓存配置——在web config文件（<system.web><sessionState>节点）以及RedisSessionStateProvider.cs文件中。

在MVC项目中有很多待办事项，因此，如果你想改进/继续，请更新，并上传。

如果有人能提供使用Redis（以及Funq IOC）缓存的MVC应用程序示例，本人将不胜感激。Funq IOC已经配置，使用示例已经在Question controller中。

注：部分取样于“ServiceStack.Examples-master”解决方案。

结论。优化应用程序缓存以及快速本地缓存

由于Redis并不在本地存储（也不在本地复制）数据，那么通过在本地缓存区存储一些轻量级或用户依赖的对象（跳过序列化字符串和客户端—服务端数据转换）来优化性能是有意义的。例如，在Web应用中，对于轻量级的对象使用’System.Runtime.Caching.ObjectCache‘ 会更好——用户依赖，并且应用程序时常要用。否则，当经常性地需要使用该对象时，就必须在分布式Redis缓存中存储大量容积的内容。用户依赖的对象举例——个人资料信息，个性化信息 。常用对象——本地化数据，不同用户之间的共享信息，等等。

[下载源代码（Redis Funq LoC MVC 4版本）](http://www.codeproject.com/KB/web-cache/636730/RedisClientSample.zip)

链接

如何运行Redis服务：

<https://github.com/kcherenkov/redis-windows-service>

文档：

<http://redis.io/documentation>

.NET / C#示例：

<https://github.com/ServiceStack/ServiceStack.Examples>

关于如何用C＃在Windows上使用Redis的好建议：

[http://maxivak.com/getting-started-with-redis-and-asp-net-mvc-under-windows/：](http://maxivak.com/getting-started-with-redis-and-asp-net-mvc-under-windows/%EF%BC%9A)

<http://www.piotrwalat.net/using-redis-with-asp-net-web-api/>

关于Redis：

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

Azure缓存

<http://kotugoroshko.blogspot.ae/2013/07/windows-azure-caching-integration.html>

许可证

这篇文章，以及任何相关的源代码和文件，依据The Code Project Open License (CPOL)。

译文链接：<http://www.codeceo.com/article/distributed-caching-redis-server.html>  
英文原文：[Distributed Caching using Redis Server with .NET/C# Client](http://www.codeproject.com/Articles/636730/Distributed-Caching-using-Redis)