Kostenloses eBook

LERNEN StackExchange.Redis

Free unaffiliated eBook created from
Stack Overflow contributors.

#stackexch

ange.redis

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: stackexchange-redis

It is an unofficial and free StackExchange.Redis ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official StackExchange.Redis.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit StackExchange.Redis

Bemerkungen

Installieren

Binaries für StackExchange.Redis sind auf Nuget verfügbar und die Quelle ist auf Github verfügbar .

Häufige Aufgaben

- Profilierung

Versionen

Ausführung	Veröffentlichungsdatum
1.0.187	2014-03-18

Examples

Grundlegende Verwendung

```
using StackExchange.Redis;

// ...

// connect to the server
ConnectionMultiplexer connection = ConnectionMultiplexer.Connect("localhost");

// select a database (by default, DB = 0)
IDatabase db = connection.GetDatabase();

// run a command, in this case a GET
RedisValue myVal = db.StringGet("mykey");
```

Multiplexer in verschiedenen Anwendungen wiederverwenden

```
class Program
{
    private static Lazy<ConnectionMultiplexer> _multiplexer =
        new Lazy<ConnectionMultiplexer>(
        () => ConnectionMultiplexer.Connect("localhost"),
        LazyThreadSafetyMode.ExecutionAndPublication);
```

```
    static void Main(string[] args)
    {
        IDatabase db1 = _multiplexer.Value.GetDatabase(1);
        IDatabase db2 = _multiplexer.Value.GetDatabase(2);
    }
}
```

Einstellmöglichkeiten

Stellen Sie eine Verbindung zum Redis-Server her, und geben Sie Administratorbefehle (riskant) an

```
ConfigurationOptions options = new ConfigurationOptions()
            {
                EndPoints = { { "localhost", 6379}},
                AllowAdmin = true,
                ConnectTimeout = 60*1000,
            };
ConnectionMultiplexer multiplexer = ConnectionMultiplexer.Connect(options);
```

oder

```
ConnectionMultiplexer multiplexer =
    ConnectionMultiplexer.Connect("localhost:6379,allowAdmin=True,connectTimeout=60000");
```

Stellen Sie über SSL eine Verbindung zum Redis-Server her

```
 ConfigurationOptions options = new ConfigurationOptions()
            {
                EndPoints = { { "localhost", 6380}},
                Ssl = true,
                Password = "12345"
            };
ConnectionMultiplexer multiplexer = ConnectionMultiplexer.Connect(options);
```

oder

```
ConnectionMultiplexer multiplexer =
     ConnectionMultiplexer.Connect("localhost:6380,ssl=True,password=12345");
```

Erste Schritte mit StackExchange.Redis online lesen: https://riptutorial.com/de/stackexchange-redis/topic/1/erste-schritte-mit-stackexchange-redis

Kapitel 2: Pipelining

Examples

Pipelining und Multiplexing

```
var multiplexer = ConnectionMultiplexer.Connect("localhost");
IDatabase db = multiplexer.GetDatabase();

// intialize key with empty string
await db.StringSetAsync("key", "");

// create transaction that utilize multiplexing and pipelining
ITransaction transacton = db.CreateTransaction();
Task<long> appendA = transacton.StringAppendAsync("key", "a");
Task<long> appendB = transacton.StringAppendAsync("key", "b");

if (await transacton.ExecuteAsync()) // sends "MULTI APPEND KEY a APPEND KEY b EXEC
// in single request to redis server
{
    // order here doesn't matter, result is always - "abc".
    // 'a' and 'b' append always together in isolation of other Redis commands
    // 'c' appends to "ab" because transaction is already executed successfully
    await appendA;
    await db.StringAppendAsync("key", "c");
    await appendB;
}

string value = db.StringGet("key"); // value is "abc"
```

Pipelining online lesen: https://riptutorial.com/de/stackexchange-redis/topic/3217/pipelining

Kapitel 3: Profilierung

Bemerkungen

Die Profilerstellungsfunktionen von `IProfiler` bestehen aus der `IProfiler` Schnittstelle und den `ConnectionMultiplexer.RegisterProfiler(IProfiler)` , `ConnectionMultiplexer.BeginProfiling(object)` und `ConnectionMultiplexer.FinishProfiling(object)` .

Beginnen und beenden Sie die Profilerstellung, um ein `object` damit verwandte Befehle zusammen gruppiert werden können.

Diese Gruppierung funktioniert, indem Sie Ihre `IProfiler` Schnittstelle zu Beginn eines Befehls nach einem Kontextobjekt abfragen, bevor Threading-Phänomene aufgetreten sind, und diesen Befehl einem beliebigen anderen Befehl zuordnen, der dasselbe Kontextobjekt enthält. Begin muss mit demselben Kontextobjekt aufgerufen werden, damit StackExchange.Redis die Profilerstellung von Befehlen mit diesem Kontextobjekt starten kann, und Finish wird aufgerufen, um die Profilerstellung zu beenden und die Ergebnisse zurückzugeben.

Examples

Gruppieren Sie alle Befehle aus einer Gruppe von Threads

```
class ToyProfiler : IProfiler
{
    public ConcurrentDictionary<Thread, object> Contexts = new ConcurrentDictionary<Thread,
object>();

    public object GetContext()
    {
        object ctx;
        if(!Contexts.TryGetValue(Thread.CurrentThread, out ctx)) ctx = null;

        return ctx;
    }
}


// ...

ConnectionMultiplexer conn = /* initialization */;
var profiler = new ToyProfiler();
var thisGroupContext = new object();

conn.RegisterProfiler(profiler);

var threads = new List<Thread>();

for (var i = 0; i < 16; i++)
{
    var db = conn.GetDatabase(i);
```

```
    var thread =
        new Thread(
            delegate()
            {
                var threadTasks = new List<Task>();

                for (var j = 0; j < 1000; j++)
                {
                    var task = db.StringSetAsync("" + j, "" + j);
                    threadTasks.Add(task);
                }

                Task.WaitAll(threadTasks.ToArray());
            }
        );

    profiler.Contexts[thread] = thisGroupContext;

    threads.Add(thread);
}

conn.BeginProfiling(thisGroupContext);

threads.ForEach(thread => thread.Start());
threads.ForEach(thread => thread.Join());

IEnumerable<IProfiledCommand> timings = conn.FinishProfiling(thisGroupContext);
```

Am Ende enthalten Timings 16.000 IProfiledCommand-Objekte - eines für jeden an redis ausgegebenen Befehl.

Gruppieren Sie Befehle basierend auf der Ausgabe von Thread

```
ConnectionMultiplexer conn = /* initialization */;
var profiler = new ToyProfiler();

conn.RegisterProfiler(profiler);

var threads = new List<Thread>();

var perThreadTimings = new ConcurrentDictionary<Thread, List<IProfiledCommand>>();

for (var i = 0; i < 16; i++)
{
    var db = conn.GetDatabase(i);

    var thread =
        new Thread(
            delegate()
            {
                var threadTasks = new List<Task>();

                conn.BeginProfiling(Thread.CurrentThread);

                for (var j = 0; j < 1000; j++)
                {
                    var task = db.StringSetAsync("" + j, "" + j);
                    threadTasks.Add(task);
                }
```

```
                Task.WaitAll(threadTasks.ToArray());

                perThreadTimings[Thread.CurrentThread] =
conn.FinishProfiling(Thread.CurrentThread).ToList();
            }
        );

    profiler.Contexts[thread] = thread;

    threads.Add(thread);
}

threads.ForEach(thread => thread.Start());
threads.ForEach(thread => thread.Join());
```

`perThreadTimings` 16 Einträge mit 1.000 IProfilingCommands-Befehlen, die vom Thread, der sie ausgegeben hat, eingegeben werden.

Profilierung online lesen: https://riptutorial.com/de/stackexchange-redis/topic/4/profilierung

Kapitel 4: Scan

Syntax

- `public IEnumerable<RedisKey> Keys(int database = 0, RedisValue pattern = default(RedisValue), int pageSize = CursorUtils.DefaultPageSize, long cursor = CursorUtils.Origin, int pageOffset = 0, CommandFlags flags = CommandFlags.None)`

Parameter

Parameter	Einzelheiten
Datenbank	Redis Datenbankindex, zu dem eine Verbindung hergestellt werden soll
Muster	*Unsicher*
Seitengröße	Anzahl der Elemente, die pro Seite zurückgegeben werden sollen
Mauszeiger	*Unsicher*
pageOffset	Anzahl der Seiten, um die die Ergebnisse ausgeglichen werden sollen
Flaggen	*Unsicher*

Bemerkungen

Der `Keys()` Aufruf wählt je nach Version des Redis-Servers entweder den Befehl `KEYS` oder den Befehl `SCAN` . Wo möglich, wird der Einsatz von `SCAN` bevorzugt, der ein `IEnumerable<RedisKey>` und nicht blockiert. `KEYS` hingegen blockieren beim Scannen des `KEYS` .

Examples

Grundlegende Überprüfung aller Schlüssel auf dem Server

```
// Connect to a target server using your ConnectionMultiplexer instance
IServer server = conn.GetServer("localhost", 6379);

// Write out each key in the server
foreach(var key in server.Keys()) {
    Console.WriteLine(key);
}
```

Iteration mit einem Cursor

```
// Connect to a target server using your ConnectionMultiplexer instance
IServer server = conn.GetServer("localhost", 6379);
```

```
var seq = server.Keys();
IScanningCursor scanningCursor = (IScanningCursor)seq;

// Use the cursor in some way...
```

Scan online lesen: https://riptutorial.com/de/stackexchange-redis/topic/66/scan

Kapitel 5: Schlüssel und Werte

Examples

Werte einstellen

Alle Werte in Redis werden letztendlich als `RedisValue` Typ gespeichert:

```
//"myvalue" here is implicitly converted to a RedisValue type
//The RedisValue type is rarely seen in practice.
db.StringSet("key", "aValue");
```

Einrichten und bekommen ein int

```
db.StringSet("key", 11021);
int i = (int)db.StringGet("key");
```

Oder mit StackExchange.Redis.Extensions :

```
db.Add("key", 11021);
int i = db.Get<int>("key");
```

Schlüssel und Werte online lesen: https://riptutorial.com/de/stackexchange-redis/topic/11/schlussel-und-werte

Kapitel 6: Veröffentlichen Abonnieren

Examples

Grundlagen

Sobald Sie verbunden sind , können Sie Nachrichten veröffentlichen, indem Sie die Methode `ISubscriber.Publish` aufrufen:

```
// grab an instance of an ISubscriber
var subscriber = connection.GetSubscriber();

// publish a message to the 'chat' channel
subscriber.Publish("chat", "This is a message")
```

Verbraucher können den Kanal mit der Methode `ISubscriber.Subscribe` abonnieren. Nachrichten, die vom Herausgeber gesendet werden, werden von dem an diese Methode übergebenen Handler behandelt.

```
// grab an instance of an ISubscriber
var subscriber = connection.GetSubscriber();

// subscribe to a messages over the 'chat' channel
subscriber.Subscribe("chat", (channel, message) => {
    // do something with the message
    Console.WriteLine((string)message);
});
```

Komplexe Daten (JSON)

Sie können komplexere Nachrichten übertragen, indem Sie die Nutzdaten vor der Veröffentlichung serialisieren:

```
// definition of a message
public class ChatMessage
{
    public Guid Id { get; set; }
    public string User { get; set; }
    public string Text { get; set; }
}

// grab an instance of an ISubscriber
var subscriber = connection.GetSubscriber();

var message = new ChatMessage
{
    Id = Guid.NewGuid(),
    User = "User 1234",
    Text = "Hello World!"
};
```

```
// serialize a ChatMessage
// this uses JIL to serialize to JSON
var json = JSON.Serialize(message);

// publish the message to the 'chat' channel
subscriber.Publish("chat", json)
```

Der Abonnent muss dann die Nachricht deserialisieren:

```
// grab an instance of an ISubscriber
var subscriber = connection.GetSubscriber();

// subscribe to messages over the 'chat' channel
subscriber.Subscribe("chat", (channel, json) => {
    var message = JSON.Deserialize<ChatMessage>(json);

    // do something with the message
    Console.WriteLine($"{message.User} said {message.Text}");
});
```

Komplexe Daten (Protobuf)

StackExchange.Redis unterstützt auch das Senden von Bytes über den Pub / Sub-Kanal. Hier verwenden wir protobuf-net, um unsere Nachricht vor dem Senden in ein Byte-Array zu serialisieren:

```
// definition of a message (marked up with Protobuf attributes)
[ProtoContract]
public class ChatMessage
{
    [ProtoMember(1)]
    public Guid Id { get; set; }
    [ProtoMember(2)]
    public string User { get; set; }
    [ProtoMember(3)]
    public string Text { get; set; }
}

// grab an instance of an ISubscriber
var subscriber = connection.GetSubscriber();

var message = new ChatMessage
{
    Id = Guid.NewGuid(),
    User = "User 1234",
    Text = "Hello World!"
};

using (var memoryStream = new MemoryStream())
{
    // serialize a ChatMessage using protobuf-net
    Serializer.Serialize(memoryStream, message);

    // publish the message to the 'chat' channel
    subscriber.Publish("chat", memoryStream.ToArray());
}
```

Wieder muss der Teilnehmer die Nachricht nach Erhalt deserialisieren:

```
// grab an instance of an ISubscriber
var subscriber = connection.GetSubscriber();

// subscribe to messages over the 'chat' channel
subscriber.Subscribe("chat", (channel, bytes) => {
    using (var memoryStream = new MemoryStream(bytes))
    {
        var message = Serializer.Deserialize<ChatMessage>(memoryStream);

        // do something with the message
        Console.WriteLine($"{message.User} said {message.Text}");
    }
});
```

Veröffentlichen Abonnieren online lesen: https://riptutorial.com/de/stackexchange-redis/topic/1610/veroffentlichen-abonnieren

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit StackExchange.Redis	Adam Lear, Community, Kevin Montrose, Nilay Vishwakarma, Vladimir Dorokhov
2	Pipelining	Vladimir Dorokhov
3	Profilierung	Jason Punyon, Kevin Montrose, Shog9
4	Scan	Joseph Vaughan
5	Schlüssel und Werte	Arie Litovsky, Cigano Morrison Mendez, Kevin Montrose
6	Veröffentlichen Abonnieren	Dean Ward