eBook Gratuit

APPRENEZ StackExchange.Redis

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#stackexch

ange.redis

Table des matières

À propos

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

Chapitre 1: Démarrer avec StackExchange.Redis

Remarques

Installation

Les binaires pour StackExchange.Redis sont disponibles sur Nuget , et la source est disponible sur Github .

Tâches communes

- Profilage

Versions

Version	Date de sortie
1.0.187	2014-03-18

Examples

Utilisation de base

```
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");
```

Réutiliser le multiplexeur à travers l'application

```
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);
    }
}
```

Options de configuration

Se connecter au serveur Redis et autoriser les commandes admin (risquées)

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

ou

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

Se connecter au serveur Redis via SSL

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

ou

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

Lire Démarrer avec StackExchange.Redis en ligne: https://riptutorial.com/fr/stackexchange-redis/topic/1/demarrer-avec-stackexchange-redis

Chapitre 2: Balayage

Syntaxe

- `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)`

Paramètres

Paramètre	Détails
base de données	Index de base de données Redis pour se connecter à
modèle	*Incertain*
taille de la page	Nombre d'éléments à renvoyer par page
le curseur	*Incertain*
pageOffset	Nombre de pages pour compenser les résultats par
drapeaux	*Incertain*

Remarques

L'appel `Keys()` sélectionne la commande `KEYS` ou `SCAN` basée sur la version du serveur Redis. Dans la mesure du possible, il préférera l'utilisation de `SCAN` qui renvoie un `IEnumerable<RedisKey>` et ne bloque pas. `KEYS` d'autre part bloquera lors du balayage de l'espace clé.

Examples

Analyse de base de toutes les clés sur le serveur

```
// 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);
}
```

Itérer en utilisant un curseur

```
// 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...
```

Lire Balayage en ligne: https://riptutorial.com/fr/stackexchange-redis/topic/66/balayage

Chapitre 3: Clés et valeurs

Examples

Définition des valeurs

Toutes les valeurs dans Redis sont finalement stockées en tant que type `RedisValue` :

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

Définir et obtenir un int

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

Ou en utilisant StackExchange.Redis.Extensions :

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

Lire Clés et valeurs en ligne: https://riptutorial.com/fr/stackexchange-redis/topic/11/cles-et-valeurs

Chapitre 4: Pipeline

Examples

Pipelining et 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"
```

Lire Pipeline en ligne: https://riptutorial.com/fr/stackexchange-redis/topic/3217/pipeline

Chapitre 5: Profilage

Remarques

Les fonctionnalités de profilage de StackExchange.Redis sont composées de l'interface `IProfiler` et des `ConnectionMultiplexer.RegisterProfiler(IProfiler)` , `ConnectionMultiplexer.BeginProfiling(object)` , `ConnectionMultiplexer.FinishProfiling(object)` .

Les profils de début et de fin prennent un `object` contexte afin que les commandes associées puissent être regroupées.

Ce regroupement fonctionne en interrogeant votre interface `IProfiler` pour un objet de contexte au début d'une commande, avant que des manipulations de thread aient eu lieu, et en associant cette commande à toute autre commande ayant le même objet de contexte. Begin doit être appelé avec le même objet de contexte afin que StackExchange.Redis sache commencer à créer des commandes de profil avec cet objet de contexte et Finish est appelé pour arrêter le profilage et retourner les résultats.

Examples

Regrouper toutes les commandes d'un ensemble de 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);
```

À la fin, les timings contiendront 16 000 objets IProfiledCommand - un pour chaque commande envoyée à redis.

Commandes de groupe basées sur l'émission de 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` se termine avec 16 entrées de 1 000 IProfilingCommands, indexées par le thread qui les a générées.

Lire Profilage en ligne: https://riptutorial.com/fr/stackexchange-redis/topic/4/profilage

Chapitre 6: Publier S'abonner

Examples

Les bases

Une fois connecté, vous pouvez publier des messages en appelant la méthode `ISubscriber.Publish` :

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

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

Les consommateurs peuvent s'abonner au canal à l'aide de la méthode `ISubscriber.Subscribe` . Les messages envoyés par l'éditeur seront gérés par le gestionnaire transmis à cette méthode.

```
// 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);
});
```

Données complexes (JSON)

Vous pouvez diffuser des messages plus complexes en sérialisant le contenu avant de le publier:

```
// 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)
```

L'abonné doit ensuite désérialiser le message:

```
// 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}");
});
```

Données complexes (Protobuf)

StackExchange.Redis supporte également l'envoi d'octets sur le canal pub / sub, ici nous utilisons protobuf-net pour sérialiser notre message à un tableau d'octets avant de l'envoyer:

```
// 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());
}
```

Encore une fois, l'abonné doit désérialiser le message dès sa réception:

```
// 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}");
    }
});
```

Lire Publier S'abonner en ligne: https://riptutorial.com/fr/stackexchange-redis/topic/1610/publier-s-abonner

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec StackExchange.Redis	Adam Lear, Community, Kevin Montrose, Nilay Vishwakarma, Vladimir Dorokhov
2	Balayage	Joseph Vaughan
3	Clés et valeurs	Arie Litovsky, Cigano Morrison Mendez, Kevin Montrose
4	Pipeline	Vladimir Dorokhov
5	Profilage	Jason Punyon, Kevin Montrose, Shog9
6	Publier S'abonner	Dean Ward