EBook Gratuito

APPRENDIMENTO

StackExchange.Redis

Free unaffiliated eBook created from
Stack Overflow contributors.

#stackexch

ange.redis

Sommario

Di

Capitolo 1: Iniziare con StackExchange.Redis

Osservazioni

Installazione

I binari per StackExchange.Redis sono disponibili su Nuget e l'origine è disponibile su Github .

Attività comuni

- profiling

Versioni

Versione	Data di rilascio
1.0.187	2014/03/18

Examples

Uso di 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");
```

Riutilizzare Multiplexer attraverso l'applicazione

```
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);
```

```
    }
}
```

Opzioni di configurazione

Collegarsi al server Redis e consentire i comandi admin (rischiosi)

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

o

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

Connessione al server Redis tramite SSL

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

o

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

Leggi Iniziare con StackExchange.Redis online: https://riptutorial.com/it/stackexchange-redis/topic/1/iniziare-con-stackexchange-redis

Capitolo 2: Chiavi e valori

Examples

Impostazione dei valori

Tutti i valori in Redis vengono infine memorizzati come tipo `RedisValue` :

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

Impostare e ottenere un int

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

O usando StackExchange.Redis.Extensions :

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

Leggi Chiavi e valori online: https://riptutorial.com/it/stackexchange-redis/topic/11/chiavi-e-valori

Capitolo 3: pipelining

Examples

Pipelining e 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"
```

Leggi pipelining online: https://riptutorial.com/it/stackexchange-redis/topic/3217/pipelining

Capitolo 4: profiling

Osservazioni

Le funzionalità di profilatura di StackExchange.Redis sono composte dall'interfaccia `IProfiler` e dai `ConnectionMultiplexer.RegisterProfiler(IProfiler)` , `ConnectionMultiplexer.BeginProfiling(object)` , `ConnectionMultiplexer.FinishProfiling(object)` .

Inizia e Finisci il profilo prendi un `object` contesto in modo che i relativi comandi possano essere raggruppati insieme.

Questo raggruppamento funziona interrogando l'interfaccia `IProfiler` per un oggetto di contesto all'inizio di un comando, prima che si verifichi qualsiasi shenanigans di threading e associando tale comando a qualsiasi altro comando che abbia lo stesso oggetto di contesto. Begin deve essere chiamato con lo stesso oggetto di contesto in modo che StackExchange.Redis sappia avviare i comandi di profilatura con tale oggetto di contesto e Fine viene chiamato per interrompere la profilazione e restituire i risultati.

Examples

Raggruppa tutti i comandi dal set di thread insieme

```
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);
```

Alla fine, i tempi conterranno 16.000 oggetti IProfiledCommand - uno per ogni comando emesso per redis.

Comandi di gruppo basati sul thread di emissione

```
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` termina con 16 voci di 1.000 IProfilingCommands, codificate dal Thread che le ha emesse.

Leggi profiling online: https://riptutorial.com/it/stackexchange-redis/topic/4/profiling

Capitolo 5: Pubblica Abbonati

Examples

Nozioni di base

Una volta connesso , puoi pubblicare messaggi chiamando il metodo `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")
```

I consumatori possono iscriversi al canale utilizzando il metodo `ISubscriber.Subscribe` . I messaggi inviati dal publisher saranno gestiti dal gestore passato a questo metodo.

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

Dati complessi (JSON)

Puoi trasmettere messaggi più complessi serializzando il payload prima di pubblicarlo:

```
// 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'abbonato deve quindi deserializzare il messaggio:

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

Dati complessi (Protobuf)

StackExchange.Redis supporta anche l'invio di byte sul pub / canale secondario, qui usiamo protobuf-net per serializzare il nostro messaggio su un array di byte prima di inviarlo:

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

Anche in questo caso l'abbonato deve deserializzare il messaggio al momento del ricevimento:

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

Leggi Pubblica Abbonati online: https://riptutorial.com/it/stackexchange-redis/topic/1610/pubblica-abbonati

Capitolo 6: Scansione

Sintassi

- `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)`

Parametri

Parametro	Dettagli
Banca dati	Indice del database Redis a cui connettersi
modello	*incerto*
dimensioni della pagina	Numero di articoli da restituire per pagina
cursore	*incerto*
pageOffset	Numero di pagine per compensare i risultati
bandiere	*incerto*

Osservazioni

La chiamata `Keys()` selezionerà il comando `KEYS` o `SCAN` base alla versione del server Redis. Ove possibile, preferirà l'uso di `SCAN` che restituisce un `IEnumerable<RedisKey>` e non blocca. `KEYS` tasti invece bloccheranno durante la scansione dello spazio dei tasti.

Examples

Scansione di base di tutti i tasti sul 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);
}
```

Iterare usando un cursore

```
// 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...
```

Leggi Scansione online: https://riptutorial.com/it/stackexchange-redis/topic/66/scansione

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con StackExchange.Redis	Adam Lear, Community, Kevin Montrose, Nilay Vishwakarma, Vladimir Dorokhov
2	Chiavi e valori	Arie Litovsky, Cigano Morrison Mendez, Kevin Montrose
3	pipelining	Vladimir Dorokhov
4	profiling	Jason Punyon, Kevin Montrose, Shog9
5	Pubblica Abbonati	Dean Ward
6	Scansione	Joseph Vaughan