EBook Gratis

APRENDIZAJE StackExchange.Redis

Free unaffiliated eBook created from
Stack Overflow contributors.

#stackexch

ange.redis

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con StackExchange.Redis

Observaciones

Instalación

Los binarios para StackExchange.Redis están disponibles en Nuget , y la fuente está disponible en Github .

Tareas comunes

- Perfilado

Versiones

Versión	Fecha de lanzamiento
1.0.187	2014-03-18

Examples

Uso básico

```
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");
```

Reutilizar multiplexor a través de la aplicación

```
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);
    }
}
```

Opciones de configuración

Conéctese al servidor de Redis y permita comandos de administración (riesgosos)

```
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");
```

Conectarse al servidor de Redis a través de 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");
```

Lea Empezando con StackExchange.Redis en línea: https://riptutorial.com/es/stackexchange-redis/topic/1/empezando-con-stackexchange-redis

Capítulo 2: Claves y valores

Examples

Valores de ajuste

Todos los valores en Redis se almacenan finalmente como un tipo `RedisValue` :

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

Configuración y obtener 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");
```

Lea Claves y valores en línea: https://riptutorial.com/es/stackexchange-redis/topic/11/claves-y-valores

Capítulo 3: Escanear

Sintaxis

- `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)`

Parámetros

Parámetro	Detalles
base de datos	Índice de base de datos Redis para conectarse a
modelo	*Inseguro*
tamaño de página	Número de artículos a devolver por página.
cursor	*Inseguro*
pageOffset	Número de páginas para compensar los resultados por
banderas	*Inseguro*

Observaciones

La llamada de `Keys()` seleccionará el comando `KEYS` o `SCAN` función de la versión del servidor Redis. Siempre que sea posible, preferirá el uso de `SCAN` que devuelve una `IEnumerable<RedisKey>` y no se bloquea. `KEYS` por otro lado, se bloquearán al escanear el espacio de la llave.

Examples

Escaneo básico de todas las claves en el servidor

```
// 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);
}
```

Iterando usando un 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...
```

Lea Escanear en línea: https://riptutorial.com/es/stackexchange-redis/topic/66/escanear

Capítulo 4: Perfilado

Observaciones

Las características de creación de perfiles de StackExchange.Redis están compuestas por la interfaz `IProfiler` y `ConnectionMultiplexer.RegisterProfiler(IProfiler)` , `ConnectionMultiplexer.BeginProfiling(object)` , `ConnectionMultiplexer.FinishProfiling(object)` .

Comenzar y finalizar el perfilado toma un `object` contexto para que los comandos relacionados puedan agruparse.

Esta agrupación funciona mediante la consulta de su interfaz de `IProfiler` para un objeto de contexto al inicio de un comando, antes de que ocurra cualquier chanchullo de subprocesos, y asociando ese comando con cualquier otro comando que tenga el mismo objeto de contexto. Se debe llamar a Begin con el mismo objeto de contexto, por lo que StackExchange.Redis sabe que debe comenzar a perfilar los comandos con ese objeto de contexto, y se llama a Finish para detener el perfilado y devolver los resultados.

Examples

Agrupa todos los comandos de un conjunto de hilos juntos

```
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);
```

Al final, los tiempos contendrán 16,000 objetos IProfiledCommand, uno para cada comando emitido para redisponer.

Agrupar comandos basados en hilo de emisión

```
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 entradas de 1,000 IProfilingCommands, codificadas por el hilo que las emitió.

Lea Perfilado en línea: https://riptutorial.com/es/stackexchange-redis/topic/4/perfilado

Capítulo 5: Publicar Suscribir

Examples

Lo esencial

Una vez conectado , puede publicar mensajes llamando al método `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")
```

Los consumidores pueden suscribirse al canal utilizando el método `ISubscriber.Subscribe` . Los mensajes enviados por el editor serán manejados por el controlador que se pasa a este método.

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

Datos complejos (JSON)

Puede transmitir mensajes más complejos si serializa la carga útil antes de publicarlo:

```
// 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)
```

El suscriptor entonces necesita deserializar el mensaje:

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

Datos complejos (Protobuf)

StackExchange.Redis también admite el envío de bytes a través del canal pub / sub, aquí usamos protobuf-net para serializar nuestro mensaje a una matriz de bytes antes de enviarlo:

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

Nuevamente, el suscriptor debe deserializar el mensaje al recibirlo:

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

Lea Publicar Suscribir en línea: https://riptutorial.com/es/stackexchange-redis/topic/1610/publicar-suscribir

Capítulo 6: Tubería

Examples

Canalización y multiplexación.

```
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"
```

Lea Tubería en línea: https://riptutorial.com/es/stackexchange-redis/topic/3217/tuberia

Creditos

S. No	Capítulos	Contributors
1	Empezando con StackExchange.Redis	Adam Lear, Community, Kevin Montrose, Nilay Vishwakarma, Vladimir Dorokhov
2	Claves y valores	Arie Litovsky, Cigano Morrison Mendez, Kevin Montrose
3	Escanear	Joseph Vaughan
4	Perfilado	Jason Punyon, Kevin Montrose, Shog9
5	Publicar Suscribir	Dean Ward
6	Tubería	Vladimir Dorokhov