RedisWorkQueue

Generated by Doxygen 1.12.0

1 RedisWorkQueue: Dotnet Implementation

A work queue, on top of a redis database, with implementations in Python, Rust, Go, Node.js (TypeScript) and Dotnet (C#).

This is the Dotnet implementations. For an overview of how the work queue works, it's limitations, and the general concepts and implementations in other languages, please read the `redis-work-queue readme`.

1.1 Documentation

Below is a brief example. More details on the core concepts can be found in the `readme`, and full API documentation can be found in `../RedisWorkQueue.pdf`.

1.2 Example Usage

```
using FreeRedis;
using RedisWorkQueue;

var redis = new RedisClient("localhost");
var workQueue = new WorkQueue("work_queue");

var item = new Item(Encoding.UTF8.GetBytes("data"), "item_1");

workQueue.AddItem(redis, item);

var queueLength = workQueue.QueueLength(redis);
Console.WriteLine($"Queue Length: {queueLength}");

var lease = workQueue.Lease(redis, 30, true, 10);
if (lease != null)
{
    Console.WriteLine($"Leased Item: {lease.ID}");
    // Do the work here
    workQueue.Complete(redis, lease);
}
else
{
    Console.WriteLine("No item available to lease");
}
```

In this example, we create a Redis client and a new instance of the `WorkQueue` class. We then add an item to the work queue using the `AddItem` method and get the length of the main queue using the `QueueLength` method.

We then try to lease an item from the work queue using the `Lease` method. If an item is available, we do the work and mark the item as completed using the `Complete` method.

Note that in this example, we pass `true` for the `block` parameter of the `Lease` method, which means the method will block and wait for an item to be available if the main queue is empty. We also pass a `timeout` value of `10` seconds, which means that if there are no items available after 10 seconds, the method will return `null`.

2 Namespace Documentation

2.1 RedisWorkQueue Namespace Reference

Classes

- class Item

  Represents an item to be stored in the Redis work queue.
- class KeyPrefix

  KeyPrefix is a string which should be prefixed to an identifier to generate a database key.
- class WorkQueue

  A work queue backed by a redis database.

3 Class Documentation

3.1 RedisWorkQueue.Item Class Reference

Represents an item to be stored in the Redis work queue.

Public Member Functions

- Item (byte[] data, string? id=null)

  Creates a new instance of the Item class with the specified data and ID.
- Item (string data, string? id=null)

  Creates a new instance of the Item class with the specified data and ID.
- T? DataJson< T > ()

  Deserializes the stored data into an object using JSON deserialization.

Static Public Member Functions

- static Item FromJson (object data, string? id=null)

  Creates a new instance of the Item class from the provided data by serializing it as JSON.

Properties

- byte[] Data `[get, set]`

  Gets or sets the data as a byte array.
- string StringData `[get, set]`

  Gets or sets the data as a (UTF-8) string.
- string ID `[get, set]`

  Gets or sets the ID of the item.

3.1.1 Detailed Description

Represents an item to be stored in the Redis work queue.

Definition at line 12 of file Item.cs.

3.1.2 Constructor & Destructor Documentation

Item() [1/2]

```
RedisWorkQueue.Item.Item (
            byte[] data,
            string?  id = null)  [inline]
```

Creates a new instance of the Item class with the specified data and ID.

Parameters

data	The data to be stored in the item.
id	An optional ID to uniquely identify the item. If not provided, a new GUID will be generated.

Definition at line 44 of file Item.cs.

Item() [2/2]

```
RedisWorkQueue.Item.Item (
            string data,
            string?  id = null)  [inline]
```

Creates a new instance of the Item class with the specified data and ID.

Parameters

data	The data to be stored in the item.
id	An optional ID to uniquely identify the item. If not provided, a new GUID will be generated.

Definition at line 57 of file Item.cs.

3.1.3 Member Function Documentation

DataJson< T >()

```
T? RedisWorkQueue.Item.DataJson< T > ()  [inline]
```

Deserializes the stored data into an object using JSON deserialization.

Template Parameters

T	The type to deserialize the data into.

Returns

The deserialized object of type T. Returns null if the deserialization fails.

Definition at line 75 of file Item.cs.

FromJson()

```
static Item RedisWorkQueue.Item.FromJson (
            object data,
            string?  id = null)  [inline], [static]
```

Creates a new instance of the Item class from the provided data by serializing it as JSON.

Parameters

data	The data to be serialized and stored in the item.
id	An optional ID to identify the item. If not provided, a new GUID will be generated.

Returns

A new instance of the Item class with the serialized JSON data.

Definition at line 65 of file Item.cs.

3.1.4 Property Documentation

Data

```
byte [] RedisWorkQueue.Item.Data  [get], [set]
```

Gets or sets the data as a byte array.

Definition at line 17 of file Item.cs.

ID

```
string RedisWorkQueue.Item.ID  [get], [set]
```

Gets or sets the ID of the item.

Definition at line 37 of file Item.cs.

StringData

```
string RedisWorkQueue.Item.StringData  [get], [set]
```

Gets or sets the data as a (UTF-8) string.

Definition at line 22 of file Item.cs.

The documentation for this class was generated from the following file:

- RedisWorkQueue/Item.cs

3.2 RedisWorkQueue.KeyPrefix Class Reference

KeyPrefix is a string which should be prefixed to an identifier to generate a database key.

Public Member Functions

- KeyPrefix (string Prefix)

  Creates a new instance of the KeyPrefix class with the specified prefix.
- string Of (string name)

  This creates the Key Prefix itself.
- KeyPrefix Concat (string name)

  Concat other onto prefix and return the result as a KeyPrefix.

Properties

- string Prefix `[get, set]`

  Gets or sets the prefix string.

3.2.1 Detailed Description

KeyPrefix is a string which should be prefixed to an identifier to generate a database key.

Definition at line 6 of file KeyPrefix.cs.

3.2.2 Constructor & Destructor Documentation

KeyPrefix()

```
RedisWorkQueue.KeyPrefix.KeyPrefix (
            string Prefix)  [inline]
```

Creates a new instance of the KeyPrefix class with the specified prefix.

Parameters

prefix	A string specifying the prefix to use for Redis keys.

Definition at line 17 of file KeyPrefix.cs.

3.2.3 Member Function Documentation

Concat()

```
KeyPrefix RedisWorkQueue.KeyPrefix.Concat (
            string name)  [inline]
```

Concat other onto prefix and return the result as a KeyPrefix.

Parameters

prefix	An instance of the KeyPrefix class representing the prefix to concatenate.
name	Name to concatenate with the prefix.

Returns

A new KeyPrefix instance with the concatenated namespaced prefix.

Definition at line 39 of file KeyPrefix.cs.

Of()

```
string RedisWorkQueue.KeyPrefix.Of (
            string name)  [inline]
```

This creates the Key Prefix itself.

Parameters

name	Name of the Redis key.

Returns

Namespaced Redis key.

Definition at line 28 of file KeyPrefix.cs.

3.2.4 Property Documentation

Prefix

```
string RedisWorkQueue.KeyPrefix.Prefix  [get], [set]
```

Gets or sets the prefix string.

Definition at line 11 of file KeyPrefix.cs.

The documentation for this class was generated from the following file:

- RedisWorkQueue/KeyPrefix.cs

3.3 RedisWorkQueue.WorkQueue Class Reference

A work queue backed by a redis database.

Public Member Functions

- WorkQueue (KeyPrefix name)

  Creates a new instance of the WorkQueue class with based on name given name.
- bool AddItem (IRedisClient db, Item item)

  Add an item to the work queue.
- void AddUniqueItem (IRedisClient db, Item item)

  Add an item, which is known to have an ID not already in the queue.
- long QueueLength (IRedisClient db)

  Gets the length of the main queue.
- long Processing (IRedisClient db)

  Gets the length of the processing queue.
- Item? Lease (IRedisClient db, int leaseSeconds, bool block, int timeout=0)

  Request a work lease from the work queue. This should be called by a worker to get work to complete.
- bool Complete (IRedisClient db, Item item)

  Marks a job as completed and remove it from the work queue. After `complete` has been called (and returns `true`), no workers will receive this job again.
- void LightClean (IRedisClient db)
- void DeepClean (IRedisClient db)

Properties

- string Session [get, set]

  Gets or sets the unique identifier for the current session.

3.3.1 Detailed Description

A work queue backed by a redis database.

Definition at line 12 of file WorkQueue.cs.

3.3.2 Constructor & Destructor Documentation

WorkQueue()

```
RedisWorkQueue.WorkQueue.WorkQueue (
            KeyPrefix name)  [inline]
```

Creates a new instance of the WorkQueue class with based on name given name.

Parameters

name	The key prefix for the work queue.

Definition at line 43 of file WorkQueue.cs.

3.3.3 Member Function Documentation

AddItem()

```
bool RedisWorkQueue.WorkQueue.AddItem (
            IRedisClient db,
            Item item)  [inline]
```

Add an item to the work queue.

If an item with the same ID already exists, this item is not added, and `false` is returned. Otherwise, if the item is added `true` is returned.

If you know the item ID is unique, and not already in the queue, use the optimised WorkQueue.AddUniqueItem instead.

Parameters

db	Redis instance.
item	Item to be added.

Definition at line 61 of file WorkQueue.cs.

AddUniqueItem()

```
void RedisWorkQueue.WorkQueue.AddUniqueItem (
            IRedisClient db,
            Item item)  [inline]
```

Add an item, which is known to have an ID not already in the queue.

Parameters

db	Redis instance.
item	Item to be added.

Definition at line 76 of file WorkQueue.cs.

Complete()

```
bool RedisWorkQueue.WorkQueue.Complete (
            IRedisClient db,
            Item item)  [inline]
```

Marks a job as completed and remove it from the work queue. After `complete` has been called (and returns `true`), no workers will receive this job again.

Parameters

db	The Redis client instance.
item	The item to be completed.

Returns

True if the item was successfully completed and removed, otherwise false.

Definition at line 177 of file WorkQueue.cs.

DeepClean()

```
void RedisWorkQueue.WorkQueue.DeepClean (
            IRedisClient db)  [inline]
```

Definition at line 214 of file WorkQueue.cs.

Lease()

```
Item?  RedisWorkQueue.WorkQueue.Lease (
            IRedisClient db,
            int leaseSeconds,
            bool block,
            int timeout = 0)  [inline]
```

Request a work lease from the work queue. This should be called by a worker to get work to complete.

When completed, the `complete` method should be called.

If `block` is true, the function will return either when a job is leased or after `timeout` seconds if `timeout` isn't 0.

If the job is not completed before the end of `leaseDuration`, another worker may pick up the same job.

It is not a problem if a job is marked as `done` more than once.

If you haven't already, it's worth reading the documentation on leasing items: https://github.com/MeVitae/redis-work-queue/blob/main/README.md#leasing-an-item

Parameters

db	The Redis client instance.
leaseSeconds	The number of seconds to lease the item for.
block	Indicates whether to block and wait for an item to be available if the main queue is empty.
timeout	The maximum time to block in seconds. If 0, there is not timeout.

Returns

The leased item, or null if no item is available.

Definition at line 135 of file WorkQueue.cs.

LightClean()

```
void RedisWorkQueue.WorkQueue.LightClean (
             IRedisClient db)  [inline]
```

Definition at line 187 of file WorkQueue.cs.

Processing()

```
long RedisWorkQueue.WorkQueue.Processing (
             IRedisClient db)  [inline]
```

Gets the length of the processing queue.

Parameters

db	Redis instance.

Returns

The length of the processing queue.

Definition at line 99 of file WorkQueue.cs.

QueueLength()

```
long RedisWorkQueue.WorkQueue.QueueLength (
             IRedisClient db)  [inline]
```

Gets the length of the main queue.

Parameters

db	Redis instance.

Returns

The length of the main queue.

Definition at line 89 of file WorkQueue.cs.

3.3.4 Property Documentation

Session

```
string RedisWorkQueue.WorkQueue.Session  [get], [set]
```

Gets or sets the unique identifier for the current session.

Definition at line 17 of file WorkQueue.cs.

The documentation for this class was generated from the following file:

- RedisWorkQueue/WorkQueue.cs

4 File Documentation

4.1 Item.cs

```
00001 using System;
00002 using System.IO;
00003 using System.Runtime.Serialization.Formatters.Binary;
00004 using System.Text;
00005 using Newtonsoft.Json;
00006
00007 namespace RedisWorkQueue
00008 {
00012     public class Item
00013     {
00017         public byte[] Data { get; set; }
00018
00022         public string StringData
00023         {
00024             get
00025             {
00026                 return Encoding.UTF8.GetString(Data);
00027             }
00028             set
00029             {
00030                 Data = Encoding.UTF8.GetBytes(value);
00031             }
00032         }
00033
00037         public string ID { get; set; }
00038
00044         public Item(byte[] data, string? id = null)
00045         {
00046             // Generate a random ID if none was passed.
00047             if (string.IsNullOrEmpty(id)) id = Guid.NewGuid().ToString();
00048             this.ID = id;
00049             this.Data = data;
00050         }
00051
00057         public Item(string data, string? id = null) : this(Encoding.UTF8.GetBytes(data), id) { }
00058
00065         public static Item FromJson(object data, string? id = null)
00066         {
00067             return new Item(JsonConvert.SerializeObject(data), id);
00068         }
00069
00075         public T? DataJson<T>()
00076         {
00077             return JsonConvert.DeserializeObject<T>(Encoding.UTF8.GetString(Data));
00078         }
00079     }
00080 }
```

4.2 KeyPrefix.cs

```
00001 namespace RedisWorkQueue
00002 {
00006     public class KeyPrefix
00007     {
00011         public string Prefix { get; set; }
00012 
00017         public KeyPrefix(string Prefix)
00018         {
00019             this.Prefix = Prefix;
00020         }
00021 
00022 
00028         public string Of(string name)
00029         {
00030             return Prefix + name;
00031         }
00032 
00039         public KeyPrefix Concat(string name)
00040         {
00041             return new KeyPrefix(this.Of(name));
00042         }
00043     }
00044 }
```

4.3 WorkQueue.cs

```
00001 using System;
00002 using System.Text;
00003 using System.Linq;
00004 
00005 using FreeRedis;
00006 
00007 namespace RedisWorkQueue
00008 {
00012     public class WorkQueue
00013     {
00017         public string Session { get; set; }
00018 
00022         private string MainQueueKey { get; set; }
00023 
00027         private string ProcessingKey { get; set; }
00028 
00032         private KeyPrefix LeaseKey { get; set; }
00033 
00037         private KeyPrefix ItemDataKey { get; set; }
00038 
00043         public WorkQueue(KeyPrefix name)
00044         {
00045             this.Session = name.Of(Guid.NewGuid().ToString());
00046             this.MainQueueKey = name.Of(":queue");
00047             this.ProcessingKey = name.Of(":processing");
00048             this.LeaseKey = name.Concat(":lease:");
00049             this.ItemDataKey = name.Concat(":item:");
00050         }
00051 
00061         public bool AddItem(IRedisClient db, Item item)
00062         {
00063             if (db.SetNx(ItemDataKey.Of(item.ID), item.Data))
00064             {
00065                 db.LPush(MainQueueKey, item.ID);
00066                 return true;
00067             }
00068             return false;
00069         }
00070 
00076         public void AddUniqueItem(IRedisClient db, Item item)
00077         {
00078             using var pipe = db.StartPipe();
00079             pipe.Set(ItemDataKey.Of(item.ID), item.Data);
00080             pipe.LPush(MainQueueKey, item.ID);
00081             pipe.EndPipe();
00082         }
00083 
00089         public long QueueLength(IRedisClient db)
00090         {
00091             return db.LLen(MainQueueKey);
00092         }
00093 
00099         public long Processing(IRedisClient db)
00100         {
00101             return db.LLen(ProcessingKey);
00102         }
```

```
00103
00110         private bool LeaseExists(IRedisClient db, string itemId)
00111         {
00112             return db.Exists(LeaseKey.Of(itemId));
00113         }
00114
00115
00135         public Item? Lease(IRedisClient db, int leaseSeconds, bool block, int timeout = 0)
00136         {
00137             for (; ; )
00138             {
00139                 object maybeItemId;
00140                 if (block)
00141                     maybeItemId = db.BRPopLPush(MainQueueKey, ProcessingKey, timeout);
00142                 else
00143                     maybeItemId = db.RPopLPush(MainQueueKey, ProcessingKey);
00144
00145                 if (maybeItemId == null)
00146                     return null;
00147
00148                 string itemId;
00149                 if (maybeItemId is byte[])
00150                     itemId = Encoding.UTF8.GetString((byte[])maybeItemId);
00151                 else if (maybeItemId is string)
00152                     itemId = (string)maybeItemId;
00153                 else
00154                     throw new Exception("item id from work queue not bytes or string");
00155
00156                 var data = db.Get<byte[]>(ItemDataKey.Of(itemId));
00157                 if (data == null)
00158                 {
00159                     if (block && timeout == 0)
00160                         continue;
00161                     return null;
00162                 }
00163
00164                 db.SetEx(LeaseKey.Of(itemId), leaseSeconds, Encoding.UTF8.GetBytes(Session));
00165
00166                 return new Item(data, itemId);
00167             }
00168         }
00169
00177         public bool Complete(IRedisClient db, Item item)
00178         {
00179             using var pipe = db.StartPipe();
00180             pipe.Del(ItemDataKey.Of(item.ID));
00181             pipe.LRem(ProcessingKey, 0, item.ID);
00182             pipe.Del(LeaseKey.Of(item.ID));
00183             var results = pipe.EndPipe();
00184             return ((long)results[0]) != 0;
00185         }
00186
00187         public void LightClean(IRedisClient db)
00188         {
00189             // A light clean only checks items in the processing queue
00190             var processing = db.LRange(ProcessingKey, 0, -1);
00191             foreach (string itemId in processing)
00192             {
00193                 // If there's no lease for the item, then it should be reset.
00194                 if (!LeaseExists(db, itemId))
00195                 {
00196                     // We also check the item actually exists before pushing it back to the main queue
00197                     if (db.Exists(ItemDataKey.Of(itemId)))
00198                     {
00199                         Console.WriteLine($"{itemId} has not lease, it will be reset");
00200                         using var pipe = db.StartPipe();
00201                         pipe.LRem(ProcessingKey, 0, itemId);
00202                         pipe.LPush(MainQueueKey, itemId);
00203                         pipe.EndPipe();
00204                     }
00205                     else
00206                     {
00207                         Console.WriteLine($"{itemId} was in the processing queue but does not exist");
00208                         db.LRem(ProcessingKey, 0, itemId);
00209                     }
00210                 }
00211             }
00212         }
00213
00214         public void DeepClean(IRedisClient db)
00215         {
00216             // A deep clean checks all data keys
00217             string[] itemDataKeys;
00218             string[] mainQueue;
00219             using (var pipe = db.StartPipe())
00220             {
00221                 pipe.Keys(ItemDataKey.Of("*"));
```

```
00222                 pipe.LRange(MainQueueKey, 0, -1);
00223                 var results = pipe.EndPipe();
00224                 itemDataKeys = (string[])results[0];
00225                 mainQueue = (string[])results[1];
00226             }
00227             var processing = db.LRange(ProcessingKey, 0, -1);
00228             foreach (string itemDataKey in itemDataKeys)
00229             {
00230                 string itemId = itemDataKey.Substring(ItemDataKey.Prefix.Length);
00231                 // If the item isn't in the queue, and there's no lease for the item, then it should
00232                 // be reset.
00233                 if (!mainQueue.Contains(itemId) && !LeaseExists(db, itemId))
00234                 {
00235                     Console.WriteLine($"{itemId} has not lease, it will be reset");
00236                     using var pipe = db.StartPipe();
00237                     pipe.LRem(ProcessingKey, 0, itemId);
00238                     pipe.LPush(MainQueueKey, itemId);
00239                     pipe.EndPipe();
00240                 }
00241             }
00242         }
00243     }
00244 }
```

Index