RedisWorkQueue

Generated by Doxygen 1.9.7

1 WorkQueue

The `WorkQueue` class represents a work queue backed by a Redis database. It provides methods to add items to the queue, lease items from the queue, and mark completed items as done.

1.0.1 Properties

- `Session`: Gets or sets the unique identifier for the current session.
- `MainQueueKey`: Gets or sets the Redis key for the main queue.
- `ProcessingKey`: Gets or sets the Redis key for the processing queue.
- `CleaningKey`: Gets or sets the Redis key for the cleaning queue.
- `LeaseKey`: Gets or sets the Redis key prefix for lease keys.

1.0.2 Methods

- `AddItem(IRedisClient db, Item item)`: Adds an item to the work queue. The `db` parameter is the Redis instance and the `item` parameter is the item to be added.
- `QueueLength(IRedisClient db)`: Gets the length of the main queue. The `db` parameter is the Redis instance.
- `Processing(IRedisClient db)`: Gets the length of the processing queue. The `db` parameter is the Redis instance.
- `LeaseExists(IRedisClient db, string itemId)`: Checks if a lease exists for the specified item ID. The `db` parameter is the Redis instance and the `itemId` parameter is the ID of the item to check.
- `Lease(IRedisClient db, int leaseSeconds, bool block, int timeout = 0)`↩
  : Requests a work lease from the work queue. This should be called by a worker to get work to complete. The `db` parameter is the Redis instance, the `leaseSeconds` parameter is the number of seconds to lease the item for, the `block` parameter indicates whether to block and wait for an item to be available if the main queue is empty, and the `timeout` parameter is the maximum time to block in milliseconds.
- `Complete(IRedisClient db, Item item)`: Marks a job as completed and removes it from the work queue. The `db` parameter is the Redis instance and the `item` parameter is the item to be completed.

1.0.3 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, 10000);
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 `10000` milliseconds, 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 (object 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 serialized data as a byte array.
- 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()

```
RedisWorkQueue.Item.Item (
            object 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 serialized and stored in the item.
id	An optional ID to uniquely identify the item. If not provided, a new GUID will be generated.

Gets or sets the serialized data as a byte array.

Definition at line 29 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 79 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 69 of file Item.cs.

3.1.4 Property Documentation

Data

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

Gets or sets the serialized 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 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.

Static Public Member Functions

- static KeyPrefix Concat (KeyPrefix prefix, 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()

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

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 38 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.
- void AddItem (IRedisClient db, Item item)

  Adds item to the work queue.
- long QueueLength (IRedisClient db)

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

  Gets the length of the processing queue.
- bool LeaseExists (IRedisClient db, string itemId)

  Checks if a lease exists for the specified item ID.
- 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. 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 `lease↩ Duration`, 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`.
- bool Complete (IRedisClient db, Item item)

  Marks a job as completed and remove it from the work queue.

Properties

- string Session `[get, set]`

  Gets or sets the unique identifier for the current session.
- string MainQueueKey `[get, set]`

  Gets or sets the Redis key for the main queue.
- string ProcessingKey `[get, set]`

  Gets or sets the Redis key for the processing queue.
- string CleaningKey `[get, set]`

  Gets or sets the Redis key for the cleaning queue.
- KeyPrefix LeaseKey `[get, set]`

  Gets or sets the Redis key prefix for lease keys.

3.3.1 Detailed Description

A work queue backed by a redis database.

Definition at line 9 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 41 of file WorkQueue.cs.

3.3.3 Member Function Documentation

AddItem()

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

Adds item to the work queue.

Parameters

db	Redis instance.
item	Item to be added.

Definition at line 56 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.

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 151 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 `lease↩ Duration`, 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 milliseconds.

Returns

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

Definition at line 113 of file WorkQueue.cs.

LeaseExists()

```
bool RedisWorkQueue.WorkQueue.LeaseExists (
            IRedisClient db,
            string itemId )  [inline]
```

Checks if a lease exists for the specified item ID.

Parameters

item↩ Id	The ID of the item to check.
db	The Redis client instance.

Returns

True if lease exists, false otherwise.

Definition at line 93 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 82 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 72 of file WorkQueue.cs.

3.3.4 Property Documentation

CleaningKey

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

Gets or sets the Redis key for the cleaning queue.

Definition at line 29 of file WorkQueue.cs.

LeaseKey

```
KeyPrefix RedisWorkQueue.WorkQueue.LeaseKey  [get], [set]
```

Gets or sets the Redis key prefix for lease keys.

Definition at line 34 of file WorkQueue.cs.

MainQueueKey

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

Gets or sets the Redis key for the main queue.

Definition at line 19 of file WorkQueue.cs.

ProcessingKey

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

Gets or sets the Redis key for the processing queue.

Definition at line 24 of file WorkQueue.cs.

Session

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

Gets or sets the unique identifier for the current session.

Definition at line 14 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

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using Newtonsoft.Json;

namespace RedisWorkQueue
{
    public class Item
    {
        public byte[] Data { get; set; }

        public string ID { get; set; }

        public Item(object Data, string? ID = null)
        {
            byte[] byteData;
            if (Data is string)
                byteData = Encoding.UTF8.GetBytes((string)Data);
            else if (!(Data is byte[]))
            {
                BinaryFormatter bf = new BinaryFormatter();
                using (var ms = new MemoryStream())
                {
                    //as long as we have full control over and know what the data is then this is okay

#pragma warning disable SYSLIB0011
                    bf.Serialize(ms, Data);
#pragma warning restore SYSLIB0011
                    byteData = ms.ToArray();
                }
            }
            else
                byteData = (byte[])Data;

            if (ID == null) ID = Guid.NewGuid().ToString();

            if (byteData == null)
                throw new Exception("item failed to serialise data to byte[]");
            this.Data = byteData;
            if (ID == null)
                throw new Exception("item failed to create ID");

            this.ID = ID;
        }

        public static Item FromJson(object data, string? id = null)
        {
            return new Item(JsonConvert.SerializeObject(data), id);
        }

        public T? DataJson<T>()
        {
            return JsonConvert.DeserializeObject<T>(Encoding.UTF8.GetString(Data));
        }
    }
}
```

4.2 KeyPrefix.cs

```
namespace RedisWorkQueue
{
    public class KeyPrefix
    {
        public string Prefix { get; set; }

        public KeyPrefix(string Prefix)
        {
            this.Prefix = Prefix;
        }


        {
            return Prefix + name;
        }

        public static KeyPrefix Concat(KeyPrefix prefix, string name)
        {
            return new KeyPrefix(prefix.Of(name));
        }
    }
}
```

4.3 WorkQueue.cs

```
using System.Text;
using FreeRedis;

namespace RedisWorkQueue
{
    public class WorkQueue
    {
        public string Session { get; set; }

        public string MainQueueKey { get; set; }

        public string ProcessingKey { get; set; }

        public string CleaningKey { get; set; }

        public KeyPrefix LeaseKey { get; set; }


        public WorkQueue(KeyPrefix name)
        {
            this.Session = name.Of(Guid.NewGuid().ToString());
            this.MainQueueKey = name.Of(":queue");
            this.ProcessingKey = name.Of(":processing");
            this.CleaningKey = name.Of(":cleaning");
            this.LeaseKey = KeyPrefix.Concat(name, ":leased_by_session:");
            this.ItemDataKey = KeyPrefix.Concat(name, ":item:");
        }

        public void AddItem(IRedisClient db, Item item)
        {
            using (var pipe = db.StartPipe())
            {
                pipe.Set(ItemDataKey.Of(item.ID), item.Data);
                pipe.LPush(MainQueueKey, item.ID);

                pipe.EndPipe();
            }
        }

        public long QueueLength(IRedisClient db)
        {
            return db.LLen(MainQueueKey);
        }

        public long Processing(IRedisClient db)
        {
            return db.LLen(ProcessingKey);
        }

        public bool LeaseExists(IRedisClient db, string itemId)
        {
            return db.Exists(LeaseKey.Of(itemId));
        }


        public Item? Lease(IRedisClient db, int leaseSeconds, bool block, int timeout = 0)
        {
            object maybeItemId;
            if (block)
            {
                maybeItemId = db.BRPopLPush(MainQueueKey, ProcessingKey, timeout);
            }
            else
            {
                maybeItemId = db.RPopLPush(MainQueueKey, ProcessingKey);
            }

            if (maybeItemId == null)
                return null;

            string itemId;
            if (maybeItemId is byte[])
                itemId = Encoding.UTF8.GetString((byte[])maybeItemId);
            else if (maybeItemId is string)
                itemId = (string)maybeItemId;
            else
                throw new Exception("item id from work queue not bytes or string");

            var data = db.Get<byte[]>(ItemDataKey.Of(itemId));
            if (data == null)
                data = new byte[0];

            db.SetEx(LeaseKey.Of(itemId), leaseSeconds, Encoding.UTF8.GetBytes(Session));

            return new Item(data, itemId);
        }

        public bool Complete(IRedisClient db, Item item)
        {
            var removed = db.LRem(ProcessingKey, 0, item.ID);

            if (removed == 0)
                return false;

            string itemId = item.ID;

            using (var pipe = db.StartPipe())
            {
                pipe.Del(ItemDataKey.Of(itemId));
                pipe.Del(LeaseKey.Of(itemId));

                pipe.EndPipe();
            }

            return true;
        }
    }
}
```

Index