Using Redis

For a Distributed Observable Collection<T>

Anthony.Abate@Gmail.com

CODE CAMP NYC

MARQUEE SPONSOR

pluralsight

hardcore dev and IT training

CODE CAMP NYC

PLATINUM SPONSOR

CODE CAMP NYC

PLATINUM SPONSOR

neuron esb

CODE CAMP NYC

PLATINUM SPONSOR

SURGE

CODE CAMP NYC

GOLD SPONSORS

twilio

QuickLearn TRAINING

redgate
ingeniously simple tools

westMONROE

What Are We Trying To Do?

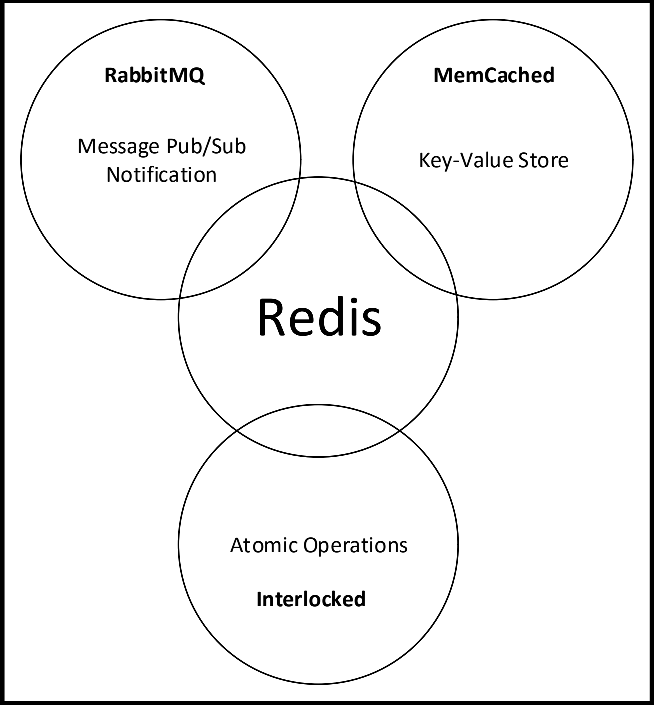
- Demonstrate some features of Redis
  - Designing a DistributedObservableCollection<T>
- Should You Do this?
  - Yes… because we can ☺
  - In all seriousness, for special low volume scenarios, sure.…
  - It won’t scale with many data, ’updates’
  - … but that is not the point

What is a Distributed Observable Collection?

- Distributed
  - Multiple Clients can read/write simultaneously
  - Notifications published to clients
- ‘Observable Collection’
  - Bindable into WPF / MVVM
  - INotifyPropertyChanged
  - OnCollectionChanged
- Concurrency
  - Row Level Versioning / Conflict Detection

Why Redis?

- Open Source (BSD License)
- Fast (ANSI-C) and In-memory Cache
- Key-Value based store
- Designed for Distributed Algorithms
- Set based operations (Data Relationships)
- Built-in Notifications Pub/Sub



Demo

- Full Featured Demonstration of Distributed Collection ‘In Action’
- As Presentation progresses, dive into the relevant code

DistributredObesrvableCollection<T> Data Requirements

- Row ViewModel
  - Unique Identifier
  - Property Storage
  - Load/Restore Initial State

- Observable Collection
  - Store Members of Collection
  - Load/Restore Initial State

DistributredObesrvableCollection<T> Notification Requirements

- Row ViewModel
  - Local Changes
    - Fire PropertyChange
    - Set in Redis
    - Publish Event
  - Remote Changes
    - Event Notification
    - Get From Redis
    - Fire PropertyChange

- Observable Collection
  - Local Changes
    - Add/Remove
    - Fire OnCollectionChanged
    - Update Redis
    - Publish Event
  - RemoteChange
    - Event Notification
    - Get Data From Redis
    - Add/Remove
    - Fire OnCollectionChanged

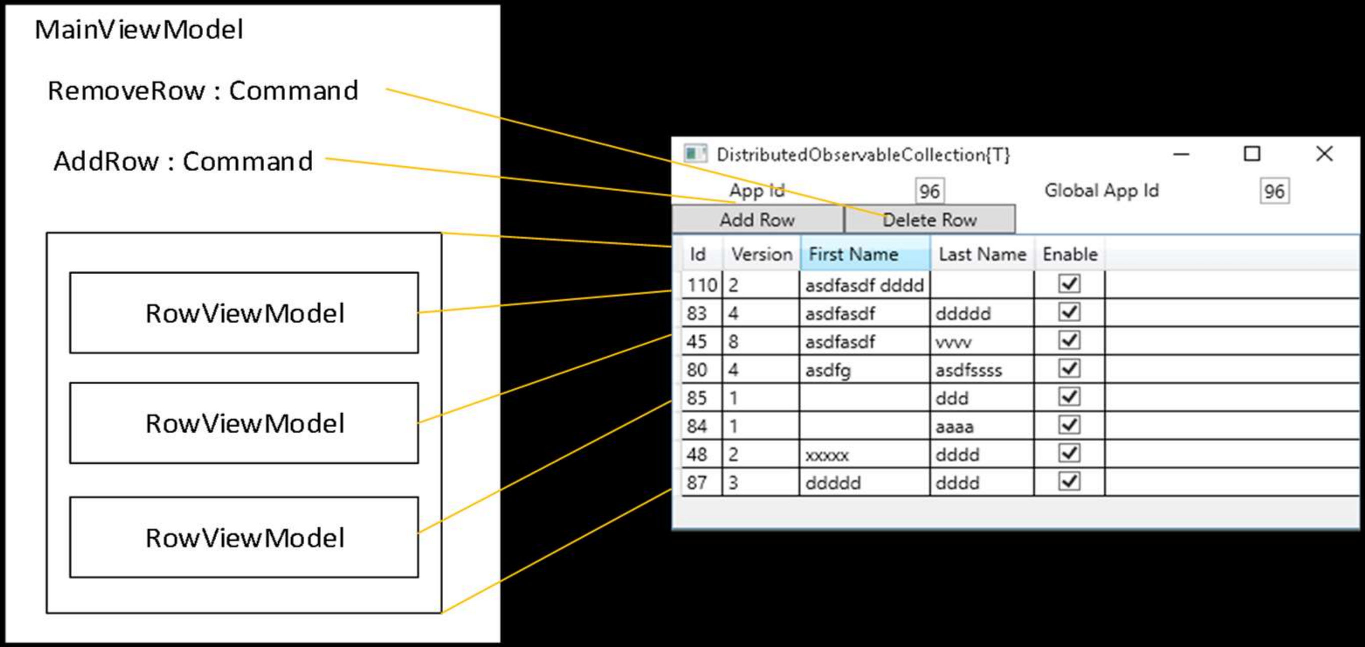
DistributredObesrvableCollection<T> Concurrency Requirements

- Row ViewModel
  - When 2 writes to the same Property are attempted, notify user of collision
  - Similar to Database row versioning

- Observable Collection
  - Set Operations are Atomic

WPF / MVVM : ViewModel <-> View

- Just an Overview



Id	Version	First Name	Last Name	Enable
110	2	asdfasdf dddd		✓
83	4	asdfasdf	ddddd	✓
45	8	asdfasdf	vvvv	✓
80	4	asdfg	asdfssss	✓
85	1		ddd	✓
84	1		aaaa	✓
48	2	xxxxx	dddd	✓
87	3	ddddd	dddd	✓

Redis High-Level Features

- Key Value Storage
- Set Operations
- Atomic Operations
- Pub/Sub Notifications
- Lua Scripts / Extensions

What Redis Is Not

- Its not a Message Queue
  - Only connected clients get messages, they do not queue up
  - Does not have message reliability / auto-redelivery / acknowledgements / etc
- It does not replace the need for long term storage with data integrity
- Not a Search / Reporting Engine
  - You are already using keys and indexes for lookups
  - Its single threaded, long running searches (SCAN operations) will slow it down

Running Redis

- Redis officially is non-windows
- But there is a windows port maintained by (MS Open Tech)
- Azure has Redis support
- Client Libraries – MANY
  - StackExchange ☺
  - ServiceStack ☹
- Many operational aspects of Redis (not shown in this presentation)
  - Clusters, Replication, disk persistence, transactions

Different Key Types

- String
- List
- Set
- Sorted Set
- Hash
- Hyperloglog

Basic Command Types

- Usually Come in Pairs
  - SET / GET
  - INCR / DECR

- [PREFIX][OPERATION][SUFFIX]
  - MSETNX
    (set multiple if none exist)
  - PSETEX
    (set expire in Milliseconds)

- PREFIX1

M	Multi
P (used with EX)	Milliseconds

- PREFIX2 – Key Type
- L – List
  S – Set
  Z – Sorted Set
  H – Hash
- SUFFIX

NX	Not Exist
EX	Expiration

Prefix/Suffix vary by key type

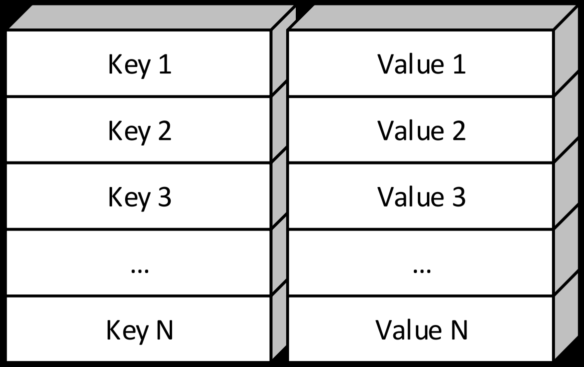
Other Command Types

- EXISTS
- DEL
- EXPIRE / EXPIREAT
- PERSIST – prevent expiration
- DUMP / RESTORE

- Searching
  - SCAN
    - Patterns / Regex
- Some Commands Can Block
  - They wait for data to become available
  - Lists – Remove Or Block Until something is available

String Key

- One Key to One Value
- Not Just For "strings"
  - Integer
  - Float
  - Bitmaps / Binary
- Max Size of 512MB

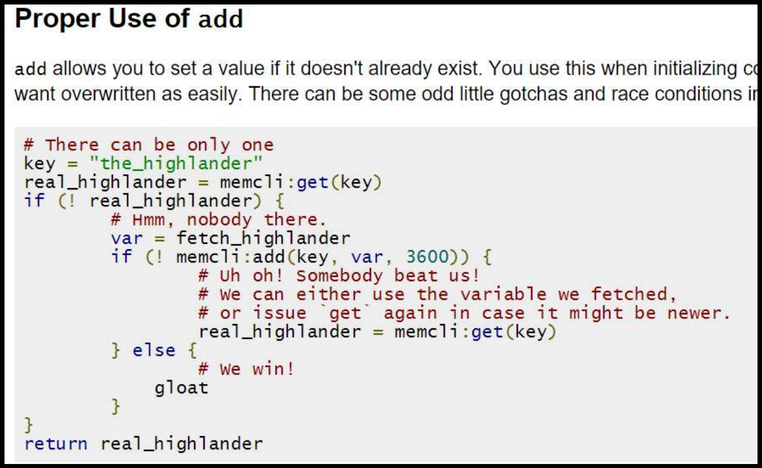


‘Some’ String Specific Operations

- String Operations
  - STRLEN
  - APPEND
  - GETRANGE (Substring)
  - SETRANGE (Replace)
- BIT Operations
  - BITCOUNT
  - BITOP
  - GETBIT / SETBIT
- GETSET – get old value / set new
- Integer/Float Operations
  - INCR/DECR
  - INCRBY/DECRBY
  - INCRBYFLOAT/DECRBYFLOAT

Atomic Operations

MemCache



Proper Use of add

add allows you to set a value if it doesn't already exist. You use this when initializing co
want overwritten as easily. There can be some odd little gotchas and race conditions ir

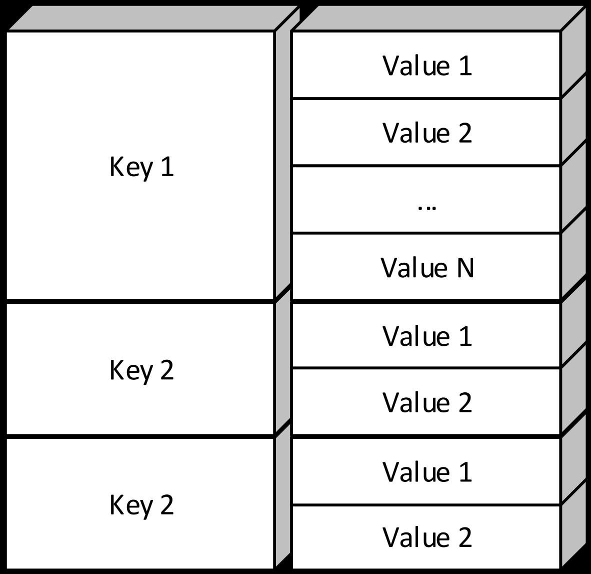
```
# There can be only one
key = "the_highlander"
real_highlander = memcli:get(key)
if (! real_highlander) {
        # Hmm, nobody there.
        var = fetch_highlander
        if (! memcli:add(key, var, 3600)) {
                # Uh oh! Somebody beat us!
                # We can either use the variable we fetched,
                # or issue `get` again in case it might be newer.
                real_highlander = memcli:get(key)
        } else {
                # We win!
            gloat
        }
}
return real_highlander
```

Redis

- INCRBY "key" 5

List Key

- One to Many Values
- “Linked List”
- Insertion Order Preserved
- Insert at Head or Tail
- Intra List Operations

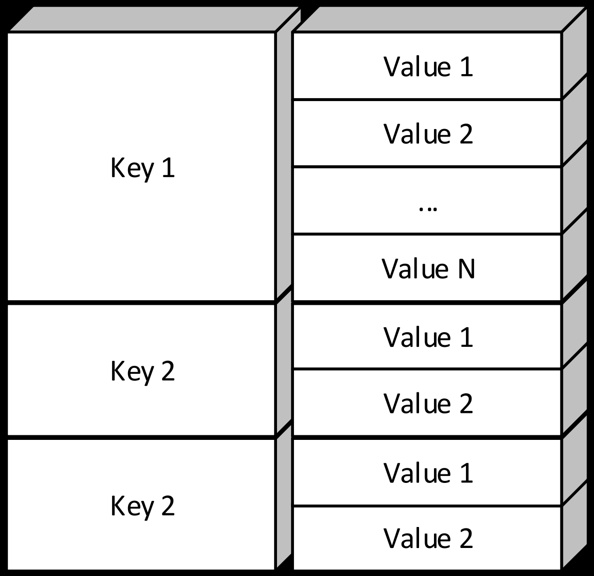


‘Some’ List Specific Operations

- Insert Operations
  - LINSERT/RINSERT
  - LPUSH,RPUSH
- Remove Operations
  - LPOP/RPOP
  - LREM/RREM
  - SPOP
- Move
  - RPOPLPUSH
  - SDIFFSTORE / SINTERSTORE / SUNIONSTORE
- Searching
  - SSCAN

Set Key

- One to Many Values
- Unordered
- Intra-Set Operations

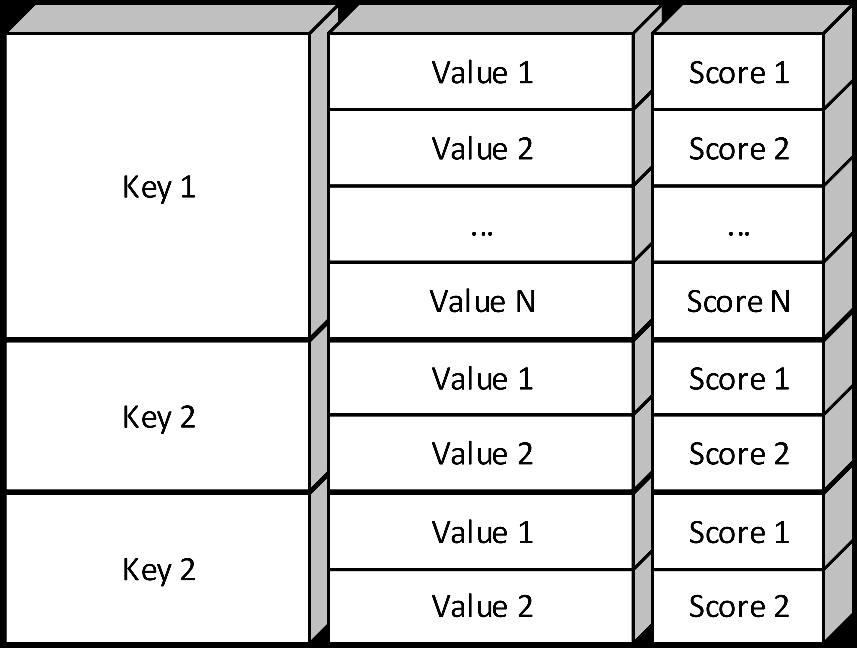


‘Some’ Set Specific Operations

- Set Operations
  - SCARD (cardinality/Size)
  - SMEMBERS
  - SISMEMBER
  - SRANDMEMER
- Changing Set
  - SADD / SREM
  - SMOVE
  - SPOP
- Intersect/Diff/Union
  - SDIFF / SINTER / SUNION
  - SDIFFSTORE / SINTERSTORE / SUNIONSTORE
- Searching
  - SSCAN

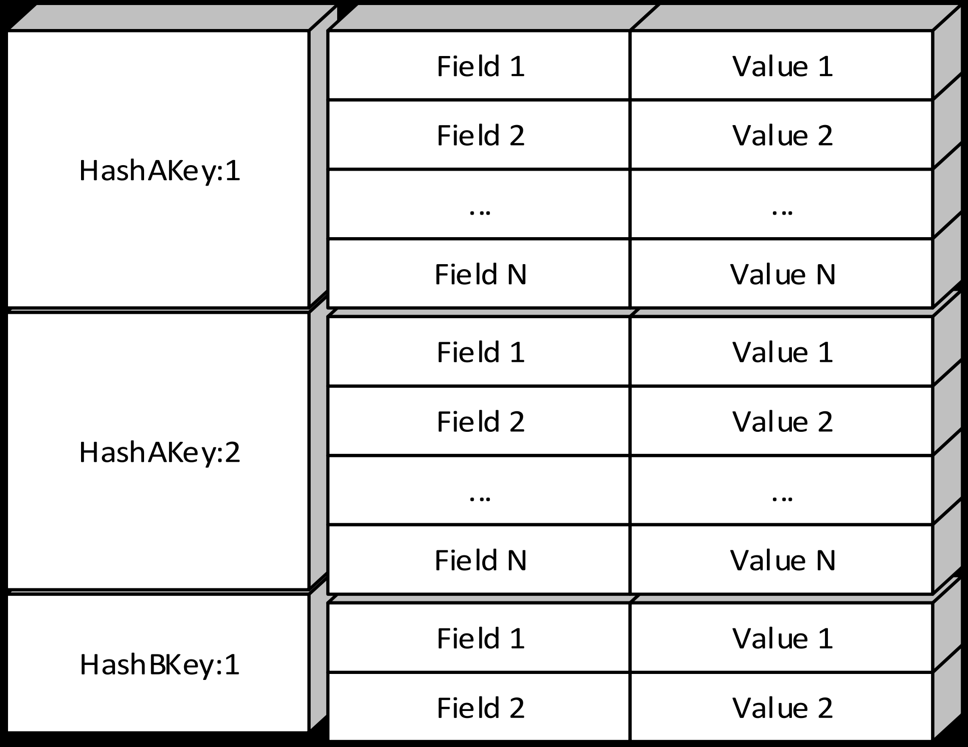
Sorted Set Key

- One Key to Many Values
- Ordered by Score / Weight
- Can update value or score
- Intra-Set Operations



Hash Key

- One Key to many Field/Values Pairs
- "HashType : ID" Naming Convention
- Each key is like a mini dictionary
  - Hash Fields have string commands
  - Atomic operations on field



Code Review / Demo

- Row ViewModel
  - Unique Identifier
  - Property Storage
  - Load/Restore Initial State

- Observable Collection
  - Store Members of Collection
  - Load/Restore Initial State

Pub / Sub - Notifications

- Simple but easy to use way of sending messages / notifications
- Up to user to create customization
- Client Libraries make it easy
- SUBSCRIBE / UNSUBSCRIBE
  - “channel”
- PUBLISH
  - “channel” “message”

Pub / Sub - Notifications

- PSUBSCRIBE / PUNSUBSCRIBE
  - Pattern Based channel name
  - h?llo -> hello, hallo and hxllo
  - h*llo -> hllo and heeeello
  - h[ae]llo -> hello and hallo, but not hillo

- PUBSUB
  - Inspect channels / subscriptions

Pub / Sub – Message Filtering

- we are publishing and receiving on the same channel
- We don’t want a feedback loop up updates
- Prefix all message with a unique client id
- Filter messages produced by the current client

Code Review / Demo

- Row ViewModel
  - Local Changes (publish)
    - Fire PropertyChange
    - Set in Redis
    - Publish Event
  - Remote Changes (subscribe)
    - Event Notification
    - Get From Redis
    - Fire PropertyChange

- Observable Collection
  - Local Changes (publish)
    - Add/Remove
    - Fire OnCollectionChanged
    - Update Redis
    - Publish Event
  - RemoteChange (subscribe)
    - Event Notification
    - Get Data From Redis
    - Add/Remove
    - Fire OnCollectionChanged

Lua Scripts

- Allow you to create custom logic executed on the Redis server
- Atomic – Only 1 script executed at a time.
- Simple C-Like style language
- Pseudo Stored Procedures
- More Like parameterized queries
- Scripts can be ‘compiled’ to save parse time and referred to by a SHA1

Lua Scripts (syntax)

- Some what strange at first:
  - 1 based array for arguments
- EVAL “SCRIPT” #Keys KEYS ARGS
  - EVAL “SCRIPT” 2 KEY1 KEY2 ARG1 ARG2 ARG3
- To execute a redis server command
  - redis.call(‘command’, ‘key’, ‘arg’)
- All Together
  - eval "return redis.call('incrby',KEYS[1], ARGV[1])" 1 newkey 5

Lua Script – Concurrecy Check

- Our version in memory = N
- user changes a field value
- Send Version # and new Field Value
- If version #'s are the same, the data wasn't modified in transit
- 'VERY SMALL' window of time for this to occur
- If versions don't match, user will be notified their change did not go through
  - They refresh and see new data
  - Can try update again.. Or maybe they decide to do something else now

Lua Script – Pseudo Code

```
if (Redis.Version = OurVersion)
        Set Field = NewValue
        return (++Version)
else
        return error
```

Lua Script – Actual Code!

```
local current = redis.call('HGET', KEYS[1], KEYS[2]);

if current == ARGV[1] then

       redis.call('HSET', KEYS[1], KEYS[3], ARGV[2])

       return redis.call('HINCRBY', KEYS[1], KEYS[2], 1)

else

       return redis.error_reply(string.format('Version Mismatch:
Expected:%s Actual:%s', ARGV[1], current))

end
```

Code Review / Demo

- Concurrency Conflict Detection