Redis guideline

Hướng dẫn triển khai Redis

- Hướng dẫn cài đặt Redis Master - Slave trên CentOS 7
- Hướng dẫn cài đặt Redis trên CentOS 7/6
- Hướng dẫn kết nối Redis sử dụng Redisson trong Java

Hướng dẫn cài đặt Redis Master - Slave trên CentOS 7

Hướng dẫn cài đặt Redis Cluster cơ chế Master - Slave trên hệ điều hành Centos 7.

1. Yêu cầu

- Hệ điều hành: CentOS 7

- Tối thiểu 3 server, trong đó:

    - Master: 192.168.0.1
    - Slave: 192.168.0.2, 192.168.0.3

- Ghi chú: Trong hướng dẫn này sẽ không bật tính năng yêu cầu xác thực của Redis, bằng cách cấu hình `protected-mode=no`

2. Cơ chế Redis Master - Slave

- Redis sentinel cung cấp một giải pháp HA cho cụm server triển khai Redis. Nghĩa là, khi có một hoặc một số Redis instance down, thì Redis của bạn vẫn hoạt động tốt.

- Redis Sentinel được thiết kế để monitor và quản lý các Redis instances bằng cách thực hiện các task sau:

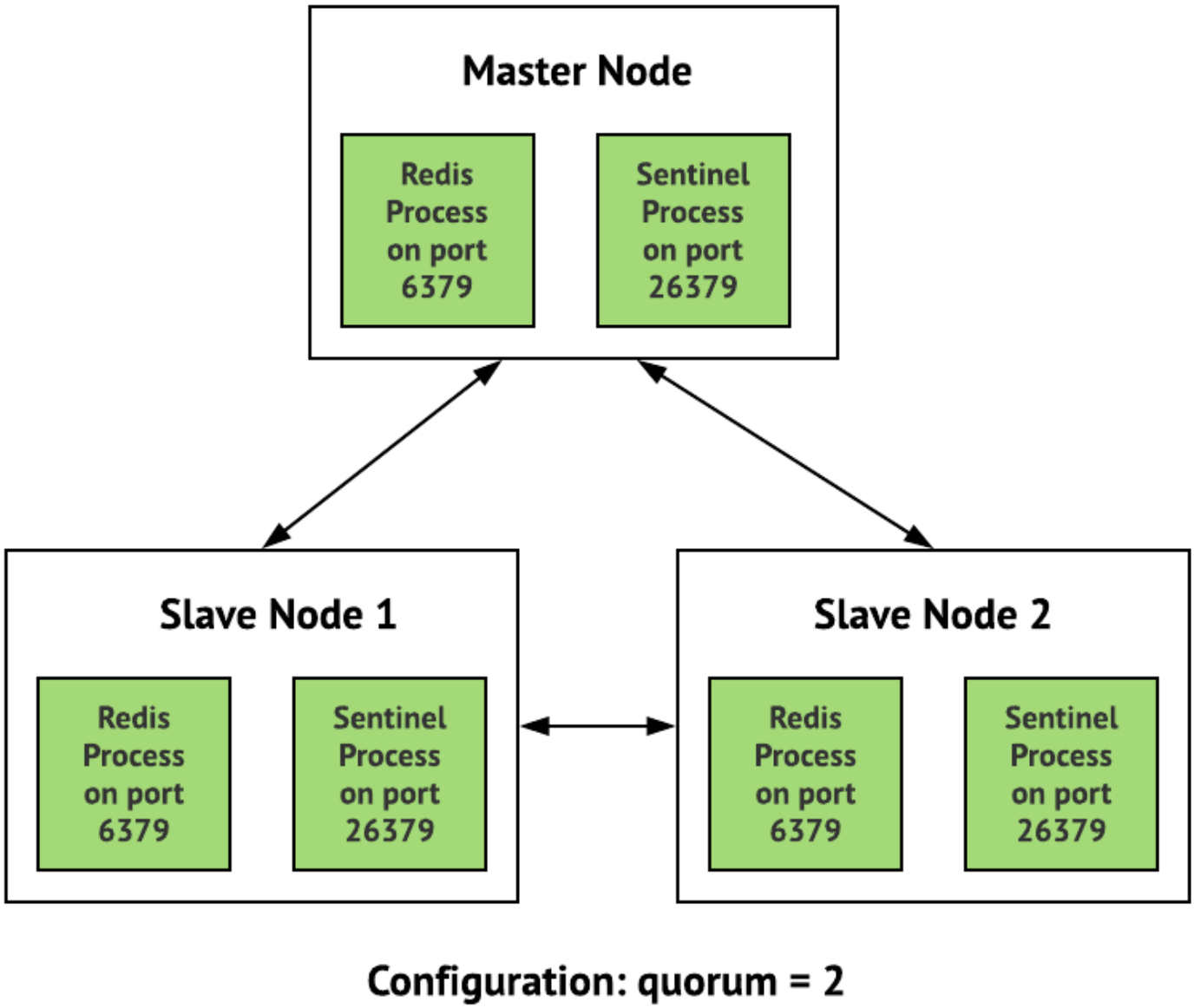
- Monitoring: Sentinel check các master và slave nodes theo định kì để đảm bảo các node này luôn luôn hoạt động tốt.
- Notification: Sentinel có thể send các notification alert khi có một redis instance failure thông tqua một API
- Automatic failover: Nếu một Master bị fail, Sentinel sẽ khởi tạo một "Failover" process:
    - Pick một running slave và promote SLAVE này lên làm Master.
    - Reconfigured các slaves còn lại thành slave của Master mới.
    - Reconfigured tất cả các Sentinel monitor Master mới.
    - Configuration provider: Sentinel hoạt động như một "source of authority for client’s service discovery". Client connect tới Sentinel để hỏi về địa chỉ của Current Redis Master, nếu một failover xảy ra, Sentinel sẽ report về cho client address của latest Redis Master (mới nhất).

- Các Sentinel liên lạc với nhau, và cùng nhau quyết định rằng một "Master node là unreachable". Quyết định được đưa ra dựa vào `quorum value` (Là số lượng Sentinel đồng ý rằng Master node là

unreachable. Giá trị này chỉ được dùng để detect failure).

- Khi Redis Master down, các Sentinel sẽ liên lạc với nhau và start một cuộc họp bàn:

- Nếu số lượng Sentinel đồng ý promote một Slave lên làm Master thì quá trình Failover sẽ được start, một Slave sẽ trở thành Master mới. Các redis instance còn lại sẽ được reconfigure thành các slave của master mới.
- Nếu số lượng Sentinel đồng ý promote một Slave lên làm Master nhỏ hơn quá bán thì quá trình Failover sẽ không diễn ra, hệ thống redis sẽ ngừng hoạt động do không có master mới.



3. Hướng dẫn cài đặt

- Cài đặt remi repo:

```
yum -y update
yum -y install http://rpms.remirepo.net/enterprise/remi-release-7.rpm
```

- Cài đặt Redis:

```
yum --enablerepo=remi install redis
rpm -qi redis
```

- Cấu hình firewall:

```
firewall-cmd --add-port=6379/tcp --permanent
firewall-cmd --reload
```

- Cấu hình sysctl:

+ Cấu hình file `/etc/sysctl.conf`

```
vm.overcommit_memory=1
```

+ Update cấu hình:

```
sysctl vm.overcommit_memory=1
```

- Cấu hình systemctl và start service:

```
systemctl enable --now redis
```

4. Cấu hình máy chủ Master

- Mở file cấu hình /etc/redis.conf và thực hiện cập nhật các cấu hình sau:

```
#bind 127.0.0.1
maxmemory-policy noeviction
tcp-keepalive 60
appendonly yes
appendfilename "appendonly.aof"
protected-mode no
```

- Restart service:

```
systemctl restart redis.service
```

5. Cấu hình máy chủ Slave

- Mở file cấu hình /etc/redis.conf và thực hiện cập nhật các cấu hình sau:

```
#bind 127.0.0.1
replicaof 192.168.0.1 6379
appendonly yes
appendfilename "appendonly.aof"
protected-mode no
```

- Restart service:

```
systemctl restart redis.service
```

6. Cấu hình Sentinel

- Mở file cấu hình /etc/redis-sentinel.conf, **xóa các cấu hình cũ và cấu hình** như sau:

```
protected-mode no
port 26379
daemonize yes
pidfile "/var/run/redis-sentinel.pid"
logfile "/var/log/redis/sentinel.log"
dir .
sentinel deny-scripts-reconfig yes
sentinel monitor mymaster 192.168.0.1 6379 2
sentinel down-after-milliseconds mymaster 5000
sentinel failover-timeout mymaster 10000
```

- Cấu hình service và start service:

```
systemctl enable --now redis-sentinel.service
```

7. Kiểm tra replication

- Trên máy chủ `master`:

```
# redis-cli
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=192.168.0.2,port=6379,state=online,offset=575099851,lag=0
slave1:ip=192.168.0.3,port=6379,state=online,offset=575099851,lag=0
master_replid:7df345aed5b6aae7512865c77ee784913d3678d7
```

```
master_replid2: d4584c75d2c5e03284e98511dba94cc095ada933
master_repl_offset: 575099851
second_repl_offset: 16516
repl_backlog_active: 1
repl_backlog_size: 1048576
repl_backlog_first_byte_offset: 574051276
repl_backlog_histlen: 1048576
```

- Trên 2 máy chủ `slave`:

```
$ redis-cli
127.0.0.1:6379> info replication
# Replication
role: slave
master_host: 192.168.0.1
master_port: 6379
master_link_status: up
master_last_io_seconds_ago: 1
master_sync_in_progress: 0
slave_repl_offset: 575120130
slave_priority: 100
slave_read_only: 1
connected_slaves: 0
master_replid: 7df345aed5b6aae7512865c77ee784913d3678d7
master_replid2: 0000000000000000000000000000000000000000
master_repl_offset: 575120130
second_repl_offset: -1
repl_backlog_active: 1
repl_backlog_size: 1048576
repl_backlog_first_byte_offset: 574071555
repl_backlog_histlen: 1048576
```

- Giờ bạn thực hiện stop redis trên máy chủ `192.168.0.1`, nếu 1 trong 2 máy chủ `192.168.0.2` hoặc `192.168.0.3` được up thành máy chủ redis master là quá trình cài đặt diễn ra thành công

Xem thêm: Hướng dẫn cài đặt Redis trên CentOS 7/6

Hướng dẫn cài đặt Redis trên CentOS 7/6

Redis là tên viết tắt của Remote Dictionary Server (Máy chủ từ điển từ xa), là một phần mềm mã nguồn mở được dùng để lưu trữ một cách tạm thời trên bộ nhớ (hay còn gọi là cache data) và giúp truy xuất dữ liệu một cách nhanh chóng. Do tốc độ truy xuất dữ liệu vượt trội, Redis thường được chọn sử dụng cho hoạt động lưu trữ bộ nhớ đệm Caching, quản lý phiên, trò chơi, bảng xếp hạng, phân tích theo thời gian thực, dữ liệu không gian địa lý, ứng dụng đặt xe, trò chuyện/nhắn tin, phát trực tiếp nội dung phương tiện và pub/sub.

1. Hướng dẫn cài đặt Redis

- Cài đặt remi repository:

```
## CentOS 7 ##
yum install epel-release
rpm -Uvh http://rpms.famillecollet.com/enterprise/remi-release-7.rpm

## CentOS 6 ##
yum install epel-release
rpm -Uvh http://rpms.famillecollet.com/enterprise/remi-release-6.rpm
```

- Cài đặt redis:

```
yum --enablerepo=remi install redis
rpm -qi redis
```

- Chạy Redis và tự động khởi động khi boot:

```
## Centos 7 ##
systemctl enable --now redis

## Centos 6 ##
chkconfig redis on
service redis start
```

2. Kiểm tra lại quá trình cài đặt Redis

- Check lại Redis server:

```
redis-cli ping
```

Nếu kết quả về `PONG` là ok

- Redis shell tools:

Mặc định Redis cài đặt với một công cụ comment là `redis-cli`

Sau khi khởi động Redis, các bạn có thể sử dụng một số command như:

- FLUSHALL – clear all databases
- SELECT # – select database under index #
- FLUSHDB – empty currently selected database
- KEYS * – list all keys from currently selected

Xem danh sách đầy đủ command ở đây.

Hướng dẫn kết nối Redis sử dụng Redisson trong Java

1. Giới thiệu

Redisson là một Redis client trong Java. Trong bài viết này, chúng ta sẽ tìm hiểu một số tính năng và demo cách xây dựng một ứng dụng sử dụng Redisson.

2. Dependency

Khai báo dependency trong maven pom.xml

```
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson</artifactId>
    <version>3.3.0</version>
</dependency>
```

3. Cấu hình

Chúng ta cần cấu hình Redisson để kết nối đến Redis. Redisson hỗ trợ các kết nối đến Redis như sau:

- Single node (Kết nối đến Redis dạng singleton) - Get single node settings here
- Master with slave nodes (Kết nối đến Redis dạng master-slave) - Get master-slave node settings here
- Sentinel nodes (Kết nối đến Redis kiểu sentinel) - Get sentinel node settings here

- Clustered nodes (Kết nối đến Redis cluster) - Get clustered node settings here
- Replicated nodes (Kết nối đến Redis replicated) - Get replicated node settings here

3.1. Java Configuration - Khai báo cấu hình trực tiếp trong Java

Chúng ta có thể cấu hình để tạo kết nối đến Redis trực tiếp trong Java như sau:

```
Config config = new Config();
config.useSingleServer()
  .setAddress("127.0.0.1:6379");

RedissonClient client = Redisson.create(config);
```

3.2. File Configuration - Khai báo cấu hình trên file

Redisson hỗ trợ load cấu hình từ file định dạng JSON hoặc YAML.

```
Config config = Config.fromJSON(new File("singleNodeConfig.json"));
RedissonClient client = Redisson.create(config);
```

Đây là mẫu config cho `singleNodeConfig.json`

```
{
    "singleServerConfig": {
        "idleConnectionTimeout": 10000,
        "pingTimeout": 1000,
        "connectTimeout": 10000,
        "timeout": 3000,
        "retryAttempts": 3,
        "retryInterval": 1500,
        "reconnectionTimeout": 3000,
        "failedAttempts": 3,
        "password": null,
        "subscriptionsPerConnection": 5,
        "clientName": null,
        "address": "redis://127.0.0.1:6379",
        "subscriptionConnectionMinimumIdleSize": 1,
        "subscriptionConnectionPoolSize": 50,
        "connectionMinimumIdleSize": 10,
        "connectionPoolSize": 64,
        "database": 0,
```

```
        "dnsMonitoring": false,
        "dnsMonitoringInterval": 5000
    },
    "threads": 0,
    "nettyThreads": 0,
    "codec": null,
    "useLinuxNativeEpoll": false
}
```

Đây là mẫu config cho `singleNodeConfig.yaml`

```
singleServerConfig:
    idleConnectionTimeout: 10000
    pingTimeout: 1000
    connectTimeout: 10000
    timeout: 3000
    retryAttempts: 3
    retryInterval: 1500
    reconnectionTimeout: 3000
    failedAttempts: 3
    password: null
    subscriptionsPerConnection: 5
    clientName: null
    address: "redis://127.0.0.1:6379"
    subscriptionConnectionMinimumIdleSize: 1
    subscriptionConnectionPoolSize: 50
    connectionMinimumIdleSize: 10
    connectionPoolSize: 64
    database: 0
    dnsMonitoring: false
    dnsMonitoringInterval: 5000
threads: 0
nettyThreads: 0
codec: !<org.redisson.codec.JsonJacksonCodec> {}
useLinuxNativeEpoll: false
```

Để lưu từ cấu hình Java sang file JSON hoặc YAML config, chúng ta thực hiện như sau:

```
Config config = new Config();
// ... we configure multiple settings here in Java
```

```
String jsonFormat = config.toJSON();
String yamlFormat = config.toYAML();
```

4. Redisson operation

Redisson hỗ trợ các interface để thực thi synchronous, asynchronous and reactive interfaces.

VD: Synchronous

```
RedissonClient client = Redisson.create();
RAtomicLong myLong = client.getAtomicLong('myLong');
```

VD: Asynchronous

```
RFuture<Boolean> isSet = myLong.compareAndSetAsync(6, 27);
isSet.handle((result, exception) -> {
    // handle the result or exception here.
});
```

VD: Reactive

```
RedissonReactiveClient client = Redisson.createReactive();
RAtomicLongReactive myLong = client.getAtomicLong("myLong");

Publisher<Boolean> isSetPublisher = myLong.compareAndSet(5, 28);
```

5. Objects

Redisson định nghĩa các Objec như sau:

- ObjectHolder
- BinaryStreamHolder
- GeospatialHolder
- BitSet
- AtomicLong
- AtomicDouble
- Topic
- BloomFilter
- HyperLogLog

5.1. ObjectHolder

ObjectHolder có thể lưu trữ tất cả các kiểu dữ liệu.

```
RBucket<Ledger> bucket = client.getBucket("ledger");
bucket.set(new Ledger());
Ledger ledger = bucket.get();
```

5.2. BinaryStreamHolder

5.3. GeospatialHolder

5.4. BitSet

5.5. AtomicLong

AtomicLong lưu trữ giá trị kiểu long

```
RAtomicLong atomicLong = client.getAtomicLong("myAtomicLong");
atomicLong.set(5);
atomicLong.incrementAndGet();
```

5.6. AtomicDouble

5.7. Topic

Topic để lưu trữ các bản tin trong cơ chế pub-sub của Redis.

```
RTopic<CustomMessage> subscribeTopic = client.getTopic("baeldung");
subscribeTopic.addListener(
  (channel, customMessage)
  -> future.complete(customMessage.getMessage()));
```

5.8. BloomFilter

5.9. HyperLogLog

6. Collections

Redisson định nghĩa và hỗ trợ các kiểu Collection như sau:

- Map
- Multimap
- Set
- SortedSet
- ScoredSortedSet
- LexSortedSet

- List
- Queue
- Deque
- BlockingQueue
- BoundedBlockingQueue
- BlockingDeque
- BlockingFairQueue
- DelayedQueue
- PriorityQueue
- PriorityDeque

6.1. Map

Redisson Map được implement từ *java.util.concurrent.ConcurrentMap* và *java.util.Map* interfaces.

Bao gồm: *RMap*, *RMapCache*, *RLocalCachedMap* và *RClusteredMap*

```
RMap<String, Ledger> map = client.getMap("ledger");
Ledger newLedger = map.put("123", new Ledger());
```

6.2. Multimap

6.3. Set

Redisson *Set* được implement từ *java.util.Set* interface.

Bao gồm: *RSet*, *RSetCache*, và *RClusteredSet*.

```
RSet<Ledger> ledgerSet = client.getSet("ledgerSet");
ledgerSet.add(new Ledger());
```

6.4. SortedSet

6.5. ScoredSortedSet

6.6. LexSortedSet

6.7. List

Redisson *Lists* được implement từ *java.util.List* interface.

Let's create a *List* with Redisson:

```
RList<Ledger> ledgerList = client.getList("ledgerList");
ledgerList.add(new Ledger());
```

6.8. Queue

6.9. Deque

6.10. BlockingQueue

6.11. BoundedBlockingQueue

6.12. BlockingDeque

6.13. BlockingFairQueue

6.14. DelayedQueue

6.15. PriorityQueue

6.16. PriorityDeque

7. Locks and Synchronizers

Redisson cho phép các thread thực hiện khóa (lock), đồng bộ (synchronization) qua applications/servers. Bao gồm:

- Lock
- FairLock
- MultiLock
- ReadWriteLock
- Semaphore
- PermitExpirableSemaphore
- CountDownLatch

7.1. Lock

Redisson's *Lock* được implement từ *java.util.concurrent.locks.Lock* interface.

```
RLock lock = client.getLock("lock");
lock.lock();
// perform some long operations...
lock.unlock();
```

7.2. FairLock

7.3. MultiLock

Redisson's *RedissonMultiLock* nhòm nhiều đối tượng kiểu *RLock* để trở thành 1 lock block:

```
RLock lock1 = clientInstance1.getLock("lock1");
RLock lock2 = clientInstance2.getLock("lock2");
RLock lock3 = clientInstance3.getLock("lock3");

RedissonMultiLock lock = new RedissonMultiLock(lock1, lock2, lock3);
lock.lock();
// perform long running operation...
lock.unlock();
```

7.4. ReadWriteLock

7.5. Semaphore

7.6. PermitExpirableSemaphore

7.7. CountDownLatch

8. Service

8.1. Remote Service

Cơ chế này hỗ trợ kiểu thực thi trên server-side từ remote method được gọi từ client-side:

Server-side register:

```
RRemoteService remoteService = client.getRemoteService();
LedgerServiceImpl ledgerServiceImpl = new LedgerServiceImpl();

remoteService.register(LedgerServiceInterface.class, ledgerServiceImpl);
```

Client-side call remote method:

```
RRemoteService remoteService = client.getRemoteService();
LedgerServiceInterface ledgerService
  = remoteService.get(LedgerServiceInterface.class);
```

```
List<String> entries = ledgerService.getEntries(10);
```

8.2. Live Object Service

9. Pipelining

Redisson hỗ trợ pipelining để thực thi một batch các lệnh:

```
RBatch batch = client.createBatch();
batch.getMap("ledgerMap").fastPutAsync("1", "2");
batch.getMap("ledgerMap").putAsync("2", "5");

List<?> result = batch.execute();
```

10. Scripting

Redisson hỗ trợ LUA scripting. Chúng ta có thể thực thi LUA scripts đến Redis:

```
client.getBucket("foo").set("bar");
String result = client.getScript().eval(Mode.READ_ONLY,
"return redis.call('get', 'foo')", RScript.ReturnType.VALUE);
```

11. Low-Level Client

Redisson hỗ trợ Low-level Client để thực thi các native Redis command:

```
RedisClient client = new RedisClient("localhost", 6379);
RedisConnection conn = client.connect();
conn.sync(StringCodec.INSTANCE, RedisCommands.SET, "test", 0);

conn.closeAsync();
client.shutdown();
```

12. Ví dụ mẫu

Xem thêm tại đây