INTRODUCTION REDIS

Maarten Smeets

09-04-2018

AMIS

INTRODUCTION REDIS

INTRODUCING REDIS

DATATYPES

PUBLISH SUBSCRIBE

PERSISTENCE

TRANSACTIONS

REJSON AND REDISEARCH

CLIENTS

INTRODUCTION REDIS

1. What is Redis
2. Why use Redis
3. Who uses Redis

INTRODUCING REDIS

- **Redis**
  First release 2009
  An open-source in-memory database project implementing a distributed, in-memory key-value store with optional durability.

INTRODUCING REDIS

redis

- Redis is an open source, in-memory data structure store
  Can be used as Database, Cache, Message broker
- NoSQL Key/Value store
- Supports multiple data structures
- Features like
  - Transactions
  - Pub/Sub
  - Scalability / availability options
  - Time to live for entries

AMIS

Mostly single threaded!
Modules can be multi threaded

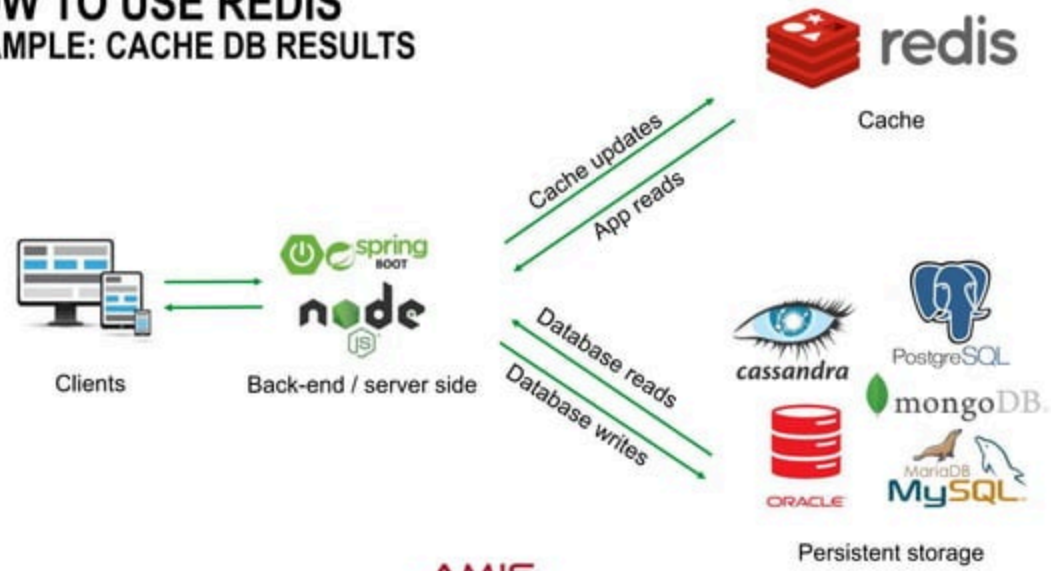
TYPICAL USES OF REDIS

- **Cache database query results**
  - https://redislabs.com/ebook/part-1-getting-started/chapter-2-anatomy-of-a-redis-web-application/2-4-database-row-caching/
- **Cache entire webpages (e.g. Wordpress)**
  - https://wordpress.org/plugins/redis-cache/
- **Use for HTTP session store**
  - https://docs.spring.io/spring-session/docs/current/reference/html5/
- **Cache logins / cookies**
  - https://redislabs.com/ebook/part-1-getting-started/chapter-2-anatomy-of-a-redis-web-application/2-1-login-and-cookie-caching/

WHY REDIS?

- Very flexible
- No schemas, column names
- Very fast
- Rich Datatype Support
- Caching & Disk persistence

HOW TO USE REDIS
EXAMPLE: CACHE DB RESULTS



WHO USES REDIS

Azure Redis Cache

AMIS

WHO USES REDIS
IN THE NETHERLANDS

kpn

=Correcthosting

Quest4

10 jaar serveo

byte

Skyberate internet services

Telfort. zakelijk

AMIS

DATATYPES

1. Strings and expiry
2. Lists, Sets, Hashes
3. Sorted sets

DATATYPES

- Strings
- Lists
- Sets
- Sorted sets
- Hashes
- Bitmaps
- Hyperlogs
- Geospatial indexes

Datatypes cannot be nested!
E.g. no lists of hashes

You can name your variables like:
person:1
person:2

KEYS person:* gives all person variables

The KEYS command is blocking.
Better to keep a set or list of person keys

AMIS

STRINGS

- Max size 512Mb
- Byte save. Can store binaries
- Use cases:
  - Store JPEG's
  - Store serialized objects

- Example usage

```
127.0.0.1:6379> SET greeting
'Hello'
OK
127.0.0.1:6379> GET greeting
"Hello"
127.0.0.1:6379> DEL greeting
(integer) 1
127.0.0.1:6379> EXISTS greeting
(integer) 0
```

EXPIRY

- Expiry can be set for existing variables

```
127.0.0.1:6379> SET greeting "Hello"
OK
127.0.0.1:6379> EXPIRE greeting 10
(integer) 1
127.0.0.1:6379> TTL greeting
(integer) 8
127.0.0.1:6379> GET greeting
"Hello"
127.0.0.1:6379> GET greeting
(nil)
```

AMIS

EXPIRY

- Expiry can be set when a variable is created and expiry can be removed

```
127.0.0.1:6379> SETEX greeting 10 "Hello"
OK
127.0.0.1:6379> TTL greeting
(integer) 8
127.0.0.1:6379> PERSIST greeting
OK
127.0.0.1:6379> TTL greeting
(integer) -1
```

LISTS

- A list of Strings
  - Max size 4294967295
- Can for example be used for
  - timelines of social networks
- Speed
  - Actions at the start or end of the list are very fast.
  - Actions in the middle are a little less fast

```
127.0.0.1:6379> LPUSH people "Maarten"
(integer) 1
127.0.0.1:6379> RPUSH people "John"
(integer) 2
127.0.0.1:6379> LRANGE people 0 -1
1) "Maarten"
2) "John"
127.0.0.1:6379> LLEN people
(integer) 2
127.0.0.1:6379> LPOP people
"Maarten"
127.0.0.1:6379> RPOP people
"John"
```

SETS

- Unordered collection of Strings
- Does not allow repeated elements
- Useful for tracking unique items
- Allows extracting random members Using SPOP, SRANDMEMBER
- Useful for intersects and diffs

```
127.0.0.1:6379> SADD cars "Honda"
(integer) 1
127.0.0.1:6379> SADD cars "Ford"
(integer) 1
127.0.0.1:6379> SADD cars "Honda"
(integer) 0
127.0.0.1:6379> SISMEMBER cars "Honda"
(integer) 1
127.0.0.1:6379> SISMEMBER cars "BMW"
(integer) 0
127.0.0.1:6379> SMEMBERS cars
1) "Ford"
2) "Honda"
127.0.0.1:6379> SMOVE cars mycars "Honda"
(integer) 1
127.0.0.1:6379> SDIFF cars mycars
1) "Ford"
```

HASHES

- Maps between string fields and values
- Ideal for storing objects

```
127.0.0.1:6379> HMSET user:1000 username antirez
password P1pp0 age 34
OK
127.0.0.1:6379> HGETALL user:1000
1) "username"
2) "antirez"
3) "password"
4) "P1pp0"
5) "age"
6) "34"
127.0.0.1:6379> HSET user:1000 password 12345
(integer) 0
127.0.0.1:6379> HGET user:1000 password
"12345"
127.0.0.1:6379> HKEYS user:1000
1) "username"
2) "password"
3) "age"
```

SORTED SETS

- Non repeating collections of Strings
- Every member has a score.
- Members are unique. Scores are not
- Ordering is from small to large score
- Ordering for items with the same score is alphabetic
- Useful for leader boards or autocomplete

```
127.0.0.1:6379> ZADD myzset 1 "one"
(integer) 1
127.0.0.1:6379> ZADD myzset 1 "one"
(integer) 0
127.0.0.1:6379> ZADD myzset 1 "uno"
(integer) 1
127.0.0.1:6379> ZADD myzset 2 "two" 3 "three"
(integer) 2
127.0.0.1:6379> ZRANGE myzset 0 -1 WITHSCORES
1) "one"
2) "1"
3) "uno"
4) "1"
5) "two"
6) "2"
7) "three"
8) "3"
127.0.0.1:6379> ZRANGE myzset 0 0
1) "one"
```

PERSISTENCE

1. Redis Database File (RDB)

2. Append Only File (OAF)

PERSISTENCE

RDB (Redis Database File)	AOF (Append Only File)
Provides point in time snapshots	Logs every write
Creates complete snapshot at specified interval	Replays at server startup. If log gets big, optimization takes place
File is in binary format	File is easily readable
On crash minutes of data can be lost	Minimal chance of data loss
Small files, fast (mostly)	Big files, 'slow'

https://www.slideshare.net/eugef/redis-persistence-in-practice-1

PUBLISH SUBSCRIBE

1. How to use it
2. Things to mind

PUBLISH SUBSCRIBE
HOW DOES IT WORK

Channel can contain glob patterns like news.*

- SUBSCRIBE [channel]
- UNSUBSCRIBE [channel]
- PUBLISH [channel] [message]

PUBLISH SUBSCRIBE
THINGS TO MIND

- A missed message has been missed permanently.
  No retry

- There is no history of messages
  Like a JMS topic without durable subscribers

TRANSACTIONS

TRANSACTIONS

- Not like relational database transactions!
- Start a transaction: MULTI
  Commands after MULTI are queued
- Execute the queued commands: EXEC
  All or none of the commands are executed
  However, if one fails, the others are still executed
- Make EXEC conditional: WATCH
  EXEC will only execute if watched variables are unchanged

There is no ZPOP but
it can be implemented like below

```
WATCH zset
element = ZRANGE zset 0 0
MULTI
ZREM zset element
EXEC
```

Redis LUA scripts are also
executed in a transaction

JSON AND SEARCH

1. ReJSON
2. RediSearch

REDIS AS A JSON STORE

JSON support can be implemented by adding the ReJSON module from Redis Labs

```
127.0.0.1:6379> JSON.SET scalar . '"Hello JSON!"'
OK
127.0.0.1:6379> JSON.SET object . '{"foo": "bar", "ans": 42}'
OK
127.0.0.1:6379> JSON.GET object
"{"foo":"bar","ans":42}"
127.0.0.1:6379> JSON.GET object .ans
"42"
```

AMIS

https://redislabs.com/blog/redis-as-a-json-store/

REDISEARCH - REDIS POWERED SEARCH ENGINE

Auto completion

```
127.0.0.1:6379> FT.SUGADD autocomplete "hello world" 100
OK
127.0.0.1:6379> FT.SUGGET autocomplete "he"
1) "hello world"
```

Full text search

```
127.0.0.1:6379> FT.ADD myIdx doc1 1.0 FIELDS title "hello" body "bla" url "http://redis.io"
OK
127.0.0.1:6379> FT.SEARCH myIdx "hello world" LIMIT 0 10
1) (integer) 1
2) "doc1"
3) 1) "title"
   2) "hello"
   3) "body"
   4) "bla"
   5) "url"
   6) "http://redis.io"
```

AMIS

http://redisearch.io/Quick_Start/
docker run -p 6379:6379 redislabs/redisearch:latest

CLIENTS

1. Node.js
2. Spring Boot

CLIENTS
NODE.JS

```
var redis = require('redis');
var client = redis.createClient(); //creates a new client

client.on('connect', function() {
        console.log('connected');
});

client.on("message", function(channel, message) {
  console.log("Message '" + message + "' on channel '" + channel + "' arrived!")
});
client.subscribe("chat");
```

Create a client and connect

Create an event handler

Subscribe to a channel

CLIENTS
SPRING BOOT

- There is no annotation available to
  - Create a container
  - Register a listener to the container
  - Register a receiver to the listener

- This requires more code than for example listening to a Kafka topic

https://spring.io/guides/gs/messaging-redis/

CLIENTS
SPRING BOOT

```
@SpringBootApplication
public class Application {

    private static final Logger LOGGER = LoggerFactory.getLogger(Application.class);

    @Bean
    RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory,
            MessageListenerAdapter listenerAdapter) {

        RedisMessageListenerContainer container = new RedisMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
        container.addMessageListener(listenerAdapter, new PatternTopic("chat"));

        return container;
    }

    @Bean
    MessageListenerAdapter listenerAdapter(Receiver receiver) {
        return new MessageListenerAdapter(receiver, "receiveMessage");
    }

    @Bean
    Receiver receiver(CountDownLatch latch) {
        return new Receiver(latch);
    }
```

```
    @Bean
    CountDownLatch latch() {
        return new CountDownLatch(1);
    }

    @Bean
    StringRedisTemplate template(RedisConnectionFactory connectionFactory) {
        return new StringRedisTemplate(connectionFactory);
    }

    public static void main(String[] args) throws InterruptedException {

        ApplicationContext ctx = SpringApplication.run(Application.class, args);

        StringRedisTemplate template = ctx.getBean(StringRedisTemplate.class);
        CountDownLatch latch = ctx.getBean(CountDownLatch.class);

        LOGGER.info("Sending message...");
        template.convertAndSend("chat", "Hello from Redis!");

        latch.await();

        System.exit(0);
    }
}
```

Wait for a single message

CLIENTS
SPRING BOOT

```
public class RedisReceiver {
    private static final Logger LOGGER = LoggerFactory.getLogger(RedisReceiver.class);

    private CountDownLatch latch;

    @Autowired
    public RedisReceiver(CountDownLatch latch) {
        this.latch = latch;
    }

    public void receiveMessage(String message) {
        LOGGER.info("Received <" + message + ">");
        latch.countDown();
    }
}
```

SOME THINGS TO MIND IN PRODUCTION

- **Redis is single threaded**
  - Want to use more CPU's? Use more Redis instances!
  - Requests are blocking. Be careful with for example KEYS commands. Mind the time complexity of operations!
- **Mind the number of connections!**
  - Implement a proxy, for example twemproxy (manages persistent connections)
- **Snapshotting is blocking**
  - Investigate persistence requirements and consider rolling BGSAVE, OAF instead of snapshotting

http://tech.trivago.com/2017/01/25/learn-redis-the-hard-way-in-production/


AMIS