REDIS

✓ 1. What is Redis?

1.Redis stands for Remote Dictionary Server.

2.It is an open-source, in-memory data structure store.

3.Can be used as a database, cache, and message broker.

✓ 2. Key Features

1.In-memory storage: Extremely fast reads/writes.

2.Persistence options: RDB snapshots, AOF (Append-Only File).

3.Data structures: Strings, Lists, Sets, Hashes, Sorted Sets, Bitmaps, HyperLogLogs, Streams.

4.Pub/Sub support for messaging.

5.Atomic operations.

6.Replication and High Availability via Redis Sentinel and Cluster.

✓ 3. Support Data Structures

redis type:-

1.Strings ***(help @string)***

```
127.0.0.1:6379> set name "lokesh"                (set key value)

OK

127.0.0.1:6379> get name                         (get key)

"lokesh"

127.0.0.1:6379> append name "lokeshkhadse"       (append key newval)

(integer) 18

127.0.0.1:6379> get name

"lokeshlokeshkhadse"
```

```
127.0.0.1:6379> incr user_id      (incr key)

(integer) 1

127.0.0.1:6379> incr user_id

(integer) 2

127.0.0.1:6379> incr user_id

(integer) 3

127.0.0.1:6379> get user_id

"3"

127.0.0.1:6379> decr user_id

(integer) 2
```

2.Hashes   (help @Hashes)

```
127.0.0.1:6379> hset student name "loki" age 21

(integer) 2

127.0.0.1:6379> hget student name

"loki"

127.0.0.1:6379> hgetall student

1) "name"

2) "loki"

3) "age"

4) "21"

127.0.0.1:6379> hdel student age

(integer) 1

127.0.0.1:6379> hgetall student

1) "name"

2) "loki"
```

```
127.0.0.1:6379> hkeys student

1) "name"

127.0.0.1:6379> hlen student

(integer) 1
```

3.list   (help @list)

```
127.0.0.1:6379> lpush marks    11 12 13 14 15

(integer) 5

127.0.0.1:6379> llen marks

(integer) 5

127.0.0.1:6379> lrange marks 0 4

1) "15"

2) "14"

3) "13"

4) "12"

5) "11"

127.0.0.1:6379> rpush marks1 30 31 32

(integer) 3

127.0.0.1:6379> llen marks2

(integer) 0

127.0.0.1:6379> llen marks1

(integer) 3

127.0.0.1:6379> lrange marks1 0 2

1) "30"

2) "31"

3) "32"
```

```
127.0.0.1:6379> linsert marks1 before 31 30.5

(integer) 4

127.0.0.1:6379> lrange marks1 0 3

1) "30"

2) "30.5"

3) "31"

4) "32"

127.0.0.1:6379> lpop marks1

"30"

127.0.0.1:6379> rpop marks1

"32"

127.0.0.1:6379> lrange marks1 0 1

1) "30.5"

2) "31"

127.0.0.1:6379> RPUSH namelist ram sham gana

(integer) 3

  127.0.0.1:6379> LRANGE namelist 0 2

1) "ram"

2) "sham"

3) "gana"

127.0.0.1:6379> lindex namelist 0

"ram"
```

4.set (help @set)

```
127.0.0.1:6379> sadd fruits apple banana mango

(integer) 3

127.0.0.1:6379> sadd fruits banana

(integer) 0

127.0.0.1:6379> smembers fruits

1) "mango"

2) "apple"

3) "banana"

127.0.0.1:6379> sismember fruits mango

(integer) 1

127.0.0.1:6379> srem fruits banana

(integer) 1

127.0.0.1:6379> scard fruits

(integer) 2

127.0.0.1:6379> spop fruits

"mango"

127.0.0.1:6379> scard fruits

(integer) 1

127.0.0.1:6379> smembers fruits

1) "apple"

127.0.0.1:6379> SADD set1 a b c

(integer) 3

127.0.0.1:6379> SADD set2 b c d

(integer) 3

127.0.0.1:6379> SINTER set1 set2

1) "c"
```

```
2) "b"

127.0.0.1:6379> SUNION set1 set2

1) "a"

2) "c"

3) "b"

4) "d"

127.0.0.1:6379> SDIFF set1 set2

1) "a"

127.0.0.1:6379> DEL fruits

(integer) 1

127.0.0.1:6379>
```

step1

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

step2

```
spring.data.redis.host=localhost                #127.0.0.1
spring.data.redis.port=6379
```

step3

```
@Bean
public RedisConnectionFactory redisConnectionFactory(){

      return new LettuceConnectionFactory();
}
```

```
@Bean
public RedisTemplate<String,Object>redisTemplate(){

        RedisTemplate<String,Object> redisTemplate = new RedisTemplate<>();

        //1.connectionfactory
        redisTemplate.setConnectionFactory(redisConnectionFactory());

        //2.key serializer
        redisTemplate.setKeySerializer(new StringRedisSerializer());

        //3.value serializer
        redisTemplate.setValueSerializer(new GenericJackson2JsonRedisSerializer());


        return redisTemplate;
}
```

step4

```
@Service
//@CacheConfig(cacheNames = "users")    // Common cache name for all methods
public class UserService {

        private static final String CACHE_NAME = "users";

        @Autowired
        private UserDao userDao;

        @Cacheable(value = CACHE_NAME,key = "#userId")
        public User getUser(String userId) {
                return userDao.getUser(userId); // hit Redis manually only if not in cache
        }

        @CachePut(value = CACHE_NAME, key = "#user.userId")
        public User save(User user) {
                return userDao.save(user); // update cache
        }
```

```
    @CacheEvict(value = CACHE_NAME, key = "#userId")
    public void delete(String userId) {
        userDao.delete(userId); // remove from cache too
    }

    @CachePut(value = CACHE_NAME, key = "#userId")
    public User update(String userId, User user) {
        return userDao.updateUser(userId, user);
    }

    @Cacheable(value = CACHE_NAME)
    public List<User> findAllUsers() {
        return userDao.findAllUsers();
    }

}
```

step5

```
@Repository
public class UserDao {

    @Autowired
    private RedisTemplate<String,Object> redisTemplate;

    private static final String KEY = "USER";    //object store under this key (user)

    //save user
    public User save(User user){

        //because we storing data in key val pair that's y we use opsForHash().put(KEY,key,val)
        redisTemplate.opsForHash().put(KEY,user.getUserId(),user);
        return user;
    }

    //getUser
    public User getUser(String userId){
        //because we fetching data in key val pair that's y we use opsForHash().get(KEY,key)
        return (User) redisTemplate.opsForHash().get(KEY,userId);

    }
```

```
//findAll
public List<User> findAllUsers() {
        Map<Object, Object> userMap = redisTemplate.opsForHash().entries(KEY);
        return userMap.values().stream()
                        .map(obj -> (User) obj)
                        .collect(Collectors.toList());
}


//delete
public void delete(String userId){
        redisTemplate.opsForHash().delete(KEY,userId);
}


//update
public User updateUser(String userId,User user){
        User getUser = (User) redisTemplate.opsForHash().get(KEY,userId);
        if (getUser == null) {
                throw new RuntimeException("User not found");
        }
        getUser.setName(user.getName());

        redisTemplate.opsForHash().put(KEY,userId,getUser);
        return getUser;
}
```