Regis hỗ trợ cache dữ liệu trong bộ nhớ chính: **RAM**

**Redis**  hỗ trợ các kiểu dl: string, list, sortedList, hashset, sorted set,…

Danh sách các lệnh trong redis: [LINSERT | Redis](https://redis.io/commands/linsert/), port mặc định: 6379

ứng dụng UI desktop xem trực quan**: redis destop management**.

* **Thường ứng dụng vào các trường hợp:**
* Thường úng dụng trong các trường hợp:

+ Lưu trữ phiên làm việc của người dùng,

+ Lưu trữ bảng xếp hạng

+ úng dụng nhắn tin

+ ứng dụng sử dụng vị trí

+ Hàng đợi

**Cài đặt + sd:**

Download bản stable tại: [Download | Redis](https://redis.io/download/)

Có thể sửa port tại file **redis.conf**

Giải nén: cd redis; tar -xf archive.tar.gz\; chmod +x \*; cd redis/src; ./redis.server.sh hoặc redis-server

**- Để kiểm tra redis đang chạy hay không:**

1: cd redis/src  
 2: ./redis-cli  
 3 ping (receive **PONG** is running)

**- Chạy redis**: ./redis-server & hoặc ./redis.server.sh &  
- Nếu muốn chạy với mode protected **no:** ./redis-server **--protected-mode no &**

Thao tác trực tiếp vs redis: **./redis-cli, sau đó có thể get/set/…  
- Để shutdown:   
 1 ./redis-cli  
 2: gõ shutdown  
 3: tra các lệnh trong cli: gõ help @server**

**- Trong ứng dụng:**

* Để thao tác với redis, ta cần 1 client: **Redisson**  của thư viện để kết nối tới server redis: thường là linux server

<**dependency**> <**groupId**>org.redisson</**groupId**> <**artifactId**>redisson</**artifactId**> <**version**>3.13.1</**version**> </**dependency**> (check [here](https://search.maven.org/search?q=g:org.redisson%20AND%20a:redisson) for newest verssion)

* Redisson hỗ trợ các loại kết nối sau đây:
* Single node
* Master with slave nodes
* Sentinel nodes
* Clustered nodes
* Replicated nodes

Redisson hỗ trợ aws elasticahe cluster and azure redis cache for clusterd and replicated nodes.

**Single node:**

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

The config param can be java code or load from file

Config:

Nếu sử dụng client là  **Redisson** sẽ có 3 file config đại diện cho **sentinel, standalone, cluster:**

<dependency>  
 <groupId>org.redisson</groupId>  
 <artifactId>redisson</artifactId>  
 <version>3.16.5</version>  
</dependency>

**Standalone: quanph20.yml**

---  
singleServerConfig:  
 idleConnectionTimeout: 10000  
 connectTimeout: 10000  
 timeout: 3000  
 retryAttempts: 3  
 retryInterval: 1500  
 password: null  
 subscriptionsPerConnection: 5  
 clientName: null  
 address: "redis://10.60.156.167:6379"  
 subscriptionConnectionMinimumIdleSize: 1  
 subscriptionConnectionPoolSize: 50  
 connectionMinimumIdleSize: 24  
 connectionPoolSize: 64  
 database: 0  
 dnsMonitoringInterval: 5000  
threads: 16  
nettyThreads: 32  
codec: !<org.redisson.codec.MarshallingCodec> {}  
transportMode: "NIO"

**Cluster:** cluster-config-thonglt4.yaml

---  
clusterServersConfig:  
 idleConnectionTimeout: 10000  
 connectTimeout: 10000  
 timeout: 3000  
 retryAttempts: 3  
 retryInterval: 1500  
 failedSlaveReconnectionInterval: 3000  
 failedSlaveCheckInterval: 60000  
 password: null  
 subscriptionsPerConnection: 5  
 clientName: null  
 loadBalancer: !<org.redisson.connection.balancer.RoundRobinLoadBalancer> {}  
 subscriptionConnectionMinimumIdleSize: 1  
 subscriptionConnectionPoolSize: 50  
 slaveConnectionMinimumIdleSize: 24  
 slaveConnectionPoolSize: 64  
 masterConnectionMinimumIdleSize: 24  
 masterConnectionPoolSize: 64  
 readMode: "SLAVE"  
 subscriptionMode: "SLAVE"  
 nodeAddresses:  
 - "redis://10.60.110.22:8011"  
 - "redis://10.60.110.22:8012"  
 - "redis://10.60.110.22:8013"  
 - "redis://10.60.110.22:8014"  
 - "redis://10.60.110.22:8015"  
 - "redis://10.60.110.22:8016"  
 scanInterval: 1000  
 pingConnectionInterval: 30000  
 keepAlive: false  
 tcpNoDelay: true  
threads: 16  
nettyThreads: 32  
codec: !<org.redisson.codec.MarshallingCodec> {}  
transportMode: "NIO"

**Factory:**

Class Client {

public RedissonClient getClusterClient() {  
 if (client2 == null) {  
 Config config;  
 try {  
 config = Config.*fromYAML*(new File("config/cluster-config-thonglt4.yaml"));  
 client2 = Redisson.*create*(config);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 return client2;  
}

public static synchronized Client getInstance() {  
 if(*instance* == null){  
 synchronized(Client.class) {  
 if(*instance* == null){  
 *instance* = new Client();  
 }  
 }  
 }  
 return *instance*;  
}}

Khi sử dụng bộ thư viện của **spring-data-redis**, sẽ cần có:   
**client là Redistemplate** để thao tác với máy chủ **redis và lettuce-core** cho việc kết nối tới redis,

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
</dependency>  
  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
</dependency>  
<!-- https://mvnrepository.com/artifact/io.lettuce/lettuce-core -->  
<dependency>  
 <groupId>io.lettuce</groupId>  
 <artifactId>lettuce-core</artifactId>  
 <version>6.1.8.RELEASE</version>  
</dependency>

**Application config:**

redis.host=10.60.156.167  
redis.port=6379

**Config:**

@Configuration  
public class RedisConfig {  
 @Value(value = "${redis.host}")  
 String redisHost;  
  
 @Value(value = "${redis.port}")  
 Integer redisPort;  
  
 @Bean  
 public LettuceConnectionFactory getLettuceConnectionFactory(){  
 return new LettuceConnectionFactory(  
 new RedisStandaloneConfiguration(redisHost, redisPort));  
 }  
  
 @Bean  
 @Primary  
 public RedisTemplate<Object, Object> getRedisTemplate(RedisConnectionFactory redisConnectionFactory){  
 RedisTemplate<Object, Object> redisTemplate = new RedisTemplate<Object, Object>();  
 redisTemplate.setConnectionFactory(redisConnectionFactory);  
 return redisTemplate;  
 }  
}

**Sử dụng mô hình CRUD để tương tác với các đối tượng cần lưu vào redis**:

@Repository  
public interface RedisRepository extends CrudRepository<Student, Integer> {  
 Optional<Student> findById(Integer id);  
}

@RedisHash(value = "student")  
public class Student implements Serializable {  
 public enum Gender{  
 *MALE*, *FEMALE*, *MIX* }  
  
 @Indexed  
 private Integer id;  
 private String name;  
 private Gender gender;  
 private boolean aOk;  
  
 public Student() {  
 }  
  
 public Student(int id, String name, Gender gender, boolean aOk) {  
 this.id = id;  
 this.name = name;  
 this.gender = gender;  
 this.aOk = aOk;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Gender getGender() {  
 return gender;  
 }  
  
 public void setGender(Gender gender) {  
 this.gender = gender;  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", gender=" + gender +  
 ", isOK=" + aOk +  
 '}';  
 }  
}

**CRUD operations:**

public void workWithPrimitivesAndObject() {  
 redisTemplate.opsForValue().set("quan", "thư", 5, TimeUnit.*SECONDS*);  
 for (int i = 0; i < 10; i++) {  
 try {  
 TimeUnit.*SECONDS*.sleep(1);  
 System.*out*.println("Get key quan: " + redisTemplate.opsForValue().get("quan"));  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void workingWithRightList() {  
 List<String> list = new ArrayList<>();  
 list.add("Quan");  
 list.add("Thư");  
 if (!redisTemplate.hasKey("list")) {  
 redisTemplate.opsForList().rightPushAll("list", list);  
 } else {  
 System.*out*.println(redisTemplate.opsForList().range("list", 0, -1));  
 redisTemplate.expire("list", 10, TimeUnit.*SECONDS*);  
 }  
  
 if (!redisTemplate.hasKey("array")) {  
 redisTemplate.opsForList().rightPushAll("array", "ele1", "ele2");  
 } else {  
 System.*out*.println(redisTemplate.opsForList().range("array", 0, -1) + ", size: " + redisTemplate.opsForList().size("array"));  
 redisTemplate.delete("array");  
 }  
 }  
  
 public void workingWithLeftPush(){  
  
 List<String> leftItems = new ArrayList<>();  
 leftItems.add("l1");  
 leftItems.add("l2");  
 leftItems.add("l3");  
  
// boolean hasLeftList = redisTemplate.hasKey("left");  
 List leftSaved = redisTemplate.opsForList().range("leftList", 0, -1);  
 String size = "";  
 if(leftSaved == null){  
 redisTemplate.opsForList().leftPushAll("leftList", leftItems);  
 }else {  
 size = redisTemplate.opsForList().size("leftList")+ "";  
  
 redisTemplate.opsForList().leftPushAll("leftList", leftItems);  
 redisTemplate.opsForList().rightPop("leftList", 1);  
 }  
  
 System.*out*.println(redisTemplate.opsForList().range("leftList", 0, -1)+ ", size: " + redisTemplate.opsForList().size("leftList"));  
 }  
  
  
 public void workingWithCrud() throws Exception {  
 Student student = new Student(1, "Quan", Student.Gender.*MALE*, true);  
 Student student1 = new Student(2, "Thư", Student.Gender.*FEMALE*, true);  
 Student student2 = new Student(3, "Ánh", Student.Gender.*FEMALE*, true);  
  
 List<Student> myList = Arrays.*asList*(student, student2, student2);  
  
 redisRepository.save(student);  
 redisRepository.save(student1);  
 redisRepository.save(student2);  
 System.*out*.println("Find all:");  
 System.*out*.println(redisRepository.findAll());  
 System.*out*.println("Find student and id = 1");  
 System.*out*.println(redisRepository.findById(1).orElseThrow(() ->new Exception("No items found!")));  
  
// redisRepository.deleteAll();  
  
 }