Redis命令大全：<http://doc.redisfans.com/>

Redis面试：<https://www.cnblogs.com/jiahaoJAVA/p/6244278.html>

Redis高可用：<https://www.cnblogs.com/huangjuncong/p/8494295.html>

Redis实际使用中的一些问题: <https://blog.csdn.net/SXML123456/article/details/107282217/>

Redis 一致性问题

雪崩和穿透问题

# Redis

## 什么是redis?

Redis 是一个基于内存的高性能key-value数据库。

## Redis支持的数据类型？

String字符串：

格式: set key value

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个键最大能存储512MB。

Hash（哈希）

格式: hmset name key1 value1 key2 value2

Redis hash 是一个键值(key=>value)对集合。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

List（列表）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

格式: lpush name value

在 key 对应 list 的头部添加字符串元素

格式: rpush name value

在 key 对应 list 的尾部添加字符串元素

格式: lrem name index

key 对应 list 中删除 count 个和 value 相同的元素

格式: llen name

返回 key 对应 list 的长度

Set（集合）

格式: sadd name value

Redis的Set是string类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

zset(sorted set：有序集合)

格式: zadd name score value

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

## 什么是Redis持久化？Redis有哪几种持久化方式？优缺点是什么？

持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。

Redis 提供了两种持久化方式:RDB（默认） 和AOF

RDB：

rdb是Redis DataBase缩写

功能核心函数rdbSave(生成RDB文件)和rdbLoad（从文件加载内存）两个函数

AOF:

Aof是Append-only file缩写

每当执行服务器(定时)任务或者函数时flushAppendOnlyFile 函数都会被调用， 这个函数执行以下两个工作

aof写入保存：

WRITE：根据条件，将 aof\_buf 中的缓存写入到 AOF 文件

SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。

存储结构:

内容是redis通讯协议(RESP )格式的命令文本存储。

比较：

1、aof文件比rdb更新频率高，优先使用aof还原数据。

2、aof比rdb更安全也更大

3、rdb性能比aof好

4、如果两个都配了优先加载AOF

刚刚上面你有提到redis通讯协议(RESP )，能解释下什么是RESP？有什么特点？（可以看到很多面试其实都是连环炮，面试官其实在等着你回答到这个点，如果你答上了对你的评价就又加了一分）

RESP 是redis客户端和服务端之前使用的一种通讯协议；

RESP 的特点：实现简单、快速解析、可读性好

## Redis 有哪些架构模式？讲讲各自的特点

### 单机版

特点：简单

问题：

1. 内存容量有限 2、处理能力有限 3、无法高可用。

## 主从复制

Redis 的复制（replication）功能允许用户根据一个 Redis 服务器来创建任意多个该服务器的复制品，其中被复制的服务器为主服务器（master），而通过复制创建出来的服务器复制品则为从服务器（slave）。 只要主从服务器之间的网络连接正常，主从服务器两者会具有相同的数据，主服务器就会一直将发生在自己身上的数据更新同步 给从服务器，从而一直保证主从服务器的数据相同。

特点：

1、master/slave 角色

2、master/slave 数据相同

3、降低 master 读压力在转交从库

问题：

无法保证高可用

没有解决 master 写的压力

## 哨兵

Redis sentinel 是一个分布式系统中监控 redis 主从服务器，并在主服务器下线时自动进行故障转移。其中三个特性：

监控（Monitoring）： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。

提醒（Notification）： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

自动故障迁移（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作。

特点：

1、保证高可用

2、监控各个节点

3、自动故障迁移

缺点：主从模式，切换需要时间丢数据

没有解决 master 写的压力

## 集群（proxy 型）：

Twemproxy 是一个 Twitter 开源的一个 redis 和 memcache 快速/轻量级代理服务器； Twemproxy 是一个快速的单线程代理程序，支持 Memcached ASCII 协议和 redis 协议。

特点：

1、多种 hash 算法：MD5、CRC16、CRC32、CRC32a、hsieh、murmur、Jenkins

2、支持失败节点自动删除

3、后端 Sharding 分片逻辑对业务透明，业务方的读写方式和操作单个 Redis 一致

缺点：增加了新的 proxy，需要维护其高可用。

failover 逻辑需要自己实现，其本身不能支持故障的自动转移可扩展性差，进行扩缩容都需要手动干预

## 集群（直连型）：

从redis 3.0之后版本支持redis-cluster集群，Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。

特点：

1、无中心架构（不存在哪个节点影响性能瓶颈），少了 proxy 层。

2、数据按照 slot 存储分布在多个节点，节点间数据共享，可动态调整数据分布。

3、可扩展性，可线性扩展到 1000 个节点，节点可动态添加或删除。

4、高可用性，部分节点不可用时，集群仍可用。通过增加 Slave 做备份数据副本

5、实现故障自动 failover，节点之间通过 gossip 协议交换状态信息，用投票机制完成 Slave到 Master 的角色提升。

缺点：

1、资源隔离性较差，容易出现相互影响的情况。

2、数据通过异步复制,不保证数据的强一致性

## redis的应用场景有哪些

1，会话缓存（最常用）

2，消息队列，

比如支付3，活动排行榜或计数

4，发布，订阅消息（消息通知）

5，商品列表，评论列表等

## Redis是单进程单线程的

redis利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销

## Redis的回收策略

volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

## Redis实际使用

特别说明：本文针对的是新版 spring boot 2.1.3，其 spring data 依赖为 spring-boot-starter-data-redis,且其默认连接池为 lettuce

​  redis 作为一个高性能的内存数据库，如果不会用就太落伍了，之前在 node.js 中用过 redis，本篇记录如何将 redis 集成到 spring boot 中。提供 redis 操作类，和注解使用 redis 两种方式。主要内容如下：

* docker 安装 redis
* springboot 集成 redis
* 编写 redis 操作类
* 通过注解使用 redis

# 安装 redis

  通过 docker 安装，docker compose 编排文件如下：

# docker-compose.yml

version: "2"

services:

redis:

container\_name: redis

image: redis:3.2.10

ports:

- "6379:6379"

  然后在docker-compose.yml所在目录使用docker-compose up -d命令，启动 redis。

# 集成 springboot

  说明：springboot 版本为 2.1.3

## 添加 maven 依赖

  只需添加spring-boot-starter-data-redis依赖即可,并排除 lettuce 依赖，然后引入 jedis 和 jedis 的依赖 commons-pool2

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

<exclusions>

<exclusion>

<groupId>io.lettuce</groupId>

<artifactId>lettuce-core</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-pool2</artifactId>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

</dependency>

## 编写 springboot 配置文件

  配置文件如下：

server:

port: 8081

servlet:

context-path: /sso

spring:

application:

name: SSO

cache:

type: redis

redis:

database: 0

host: 192.168.226.5

port: 6379

# 有密码填密码，没有密码不填

password:

# 连接超时时间（ms)

timeout: 1000ms

# 高版本springboot中使用jedis或者lettuce

jedis:

pool:

# 连接池最大连接数（负值表示无限制）

max-active: 8

# 连接池最大阻塞等待时间（负值无限制)

max-wait: 5000ms

# 最大空闲链接数

max-idle: 8

# 最小空闲链接数

min-idle: 0

## 编写配置类

  配置类代码如下：

@EnableCaching//开启缓存

@Configuration

public class RedisConfig extends CachingConfigurerSupport {

/\*\*

\* 设置缓存管理器，这里可以配置默认过期时间等

\*

\* @param connectionFactory 连接池

\* @return

\*/

@Bean

public CacheManager cacheManager(RedisConnectionFactory connectionFactory) {

RedisCacheConfiguration redisCacheConfiguration = RedisCacheConfiguration

.defaultCacheConfig()

.entryTtl(Duration.ofSeconds(60));

//注意：请勿使用先new 配置对象，然后在调用entryTtl方法的方式来操作

//会导致配置不生效，原因是调用.entryTtl方法会返回一个新的配置对象，而不是在原来的配置对象上修改

RedisCacheWriter redisCacheWriter = RedisCacheWriter.nonLockingRedisCacheWriter(connectionFactory);

RedisCacheManager manager = new RedisCacheManager(redisCacheWriter, redisCacheConfiguration);

return manager;

}

@SuppressWarnings("all")

@Bean

public RedisTemplate<String, String> redisTemplate(JedisConnectionFactory factory) {

StringRedisTemplate template = new StringRedisTemplate(factory);

Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);

ObjectMapper om = new ObjectMapper();

om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);

om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON\_FINAL);

jackson2JsonRedisSerializer.setObjectMapper(om);

RedisSerializer stringSerializer = new StringRedisSerializer();

template.setKeySerializer(stringSerializer);

template.setValueSerializer(jackson2JsonRedisSerializer);

template.setHashKeySerializer(stringSerializer);

template.setHashValueSerializer(jackson2JsonRedisSerializer);

template.afterPropertiesSet();

return template;

}

//使用jedis连接池建立jedis连接工厂

@Bean

public JedisConnectionFactory jedisConnectionFactory() {

logger.info("jedisConnectionFactory:初始化了");

JedisPoolConfig config = new JedisPoolConfig();

config.setMaxIdle(maxIdle);

config.setMinIdle(minIdle);

config.setMaxWaitMillis(maxWaitMillis);

config.setMaxTotal(maxActive);

//链接耗尽时是否阻塞，默认true

config.setBlockWhenExhausted(true);

//是否启用pool的jmx管理功能，默认true

config.setJmxEnabled(true);

JedisConnectionFactory factory = new JedisConnectionFactory();

factory.setPoolConfig(config);

factory.setHostName(host);

factory.setPort(port);

factory.setPassword(password);

factory.setDatabase(database);

factory.setTimeout(timeout);

return factory;

}

}

## 使用方法

  有两种方法来进行缓存操作，一种是在方法上添加缓存注解实现各种操作，一种是手动控制。个人比较喜欢手动控制，觉得这样都在自己的掌控中。

### 通过注解使用

  主要有以下 5 个注解：

* @CacheConfig: 类级别缓存，设置缓存 key 前缀之类的
* @Cacheable: 触发缓存入口
* @CacheEvict: 触发移除缓存
* @CachePut: 更新缓存
* @Caching: 组合缓存

#### @CacheConfig

  该注解可以将缓存分类，它是类级别注解，主要用于给某个类的缓存全局配置，例子如下：

@CacheConfig(cacheNames = "redis\_test")

@Service

public class RedisService {

//....

}

上面 CacheConfig 会给类下通过注解生成的 key 加上 redis\_test 的前缀。

#### @Cacheable

  方法级别注解，根据 key 查询缓存:

* 如果 key 不存在，将方法返回值缓存到 redis 中
* 如果 key 存在，直接从缓存中取值  
  例子如下：

/\*\*

\* 缓存时间，首次查询后会缓存结果,key中的值可使用表达式计算.

\* 如不提供key，将使用默认key构造方法生成一个key

\* @return long

\*/

@Cacheable(key = "'currentTime'")

public long getTime() {

return System.currentTimeMillis();

}

多次调用此段代码会发现每次返回的值都是一样的。

#### CachePut

  用于更新缓存，每次调用都会想 db 请求，缓存数据

* 如果 key 存在，更新内容
* 如果 key 不存在，插入内容

代码如下：

/\*\*

\* 一般用于更新查插入操作，每次都会请求db

\*/

@CachePut(key = "'currentTime'+#id")

public long updateTime(String id) {

return System.currentTimeMillis();

}

每次调用此方法都会根据 key 刷新 redis 中的缓存数据。

#### @CacheEvict

  根据 key 删除缓存中的数据。allEntries=true 表示删除缓存中所有数据。  
代码如下：

@CacheEvict(key = "'currentTime'+#id",allEntries=false)

public void deleteTime(String id) {

}

#### @Caching

  本注解可将其他注解组合起来使用。比如下面的例子：

//value属性为key指定前缀

@Caching(put = {@CachePut(value = "user", key = "'name\_'+#user.name"),

@CachePut(value = "user", key = "'pass\_'+#user.password")})

public User testCaching(User user) {

return user;

}

上面的代码执行后将在 redis 中插入两条记录。使用keys \*将看到如下结果：

### 手动控制

  手动控制就相当于 mybatis 的手写 sql 语句，需要调用redisTemplate中的各种方法来进行缓存查询，缓存更新，缓存删除等操作。

  使用方法参见 util/RedisUtil 中的方法。redisTemplate基本可以实现所有的 redis 操作。



