# Redis

# Redis简介

|  |
| --- |
| Redis是一种基于**键值对**的**NoSQL**数据库，与很多键值对数据库不同的是，Redis中的值可以是String(字符串),hash(哈希),List(列表),set(集合),zset(有序集合)等多种数据结构。  redis vs memcached  redis的数据类型更丰富  redis支持持久化  Redis的特点：  **高性能：**  Redis将所有数据都存储在**内存中**，所以它的读写性能非常之高，官方的数据是可以达到10万/秒  **可靠性：**  Redis还将内存中的数据利用快照和日志的形式保存到硬盘中，这样就可以避免发生断点或机器故障时，内存数据丢失的问题。  **数据类型丰富：**  **字符串，hash，List，Set，Zset**  **来自百度百科的介绍：**  Redis是一个开源的使用ANSI [C语言](http://baike.baidu.com/view/1219.htm)编写、支持网络、可基于内存亦可持久化的日志型、Key-Value[数据库](http://baike.baidu.com/view/1088.htm)，并提供多种语言的API。从2010年3月15日起，Redis的开发工作由VMware主持。从2013年5月开始，Redis的开发由Pivotal赞助。  **对redis支持的语言有：**    应用的公司：  国外：twitter  国内：新浪微博，阿里巴巴，腾讯，百度，搜狐等等 |

# Redis应用场景

|  |
| --- |
| **缓存**  几乎所有的大型网站都会使用到缓存，合理使用缓存可以加快网站响应速度，而且可以减少对数据库的访问压力。  **计数器应用**  比如网站商品的浏览数，微博的点赞数，视频的播放次数，如果这些需要保证数据的实时性，每一次的操作都要做数据库的对应更新操作，那将会给数据库的性能带来极大的挑战。Redis支持的计数功能可以很好解决这个问题  **保存用户凭证**  实现多系统之间的单点登录凭证  **消息队列功能**  Redis提供了发布订阅功能和阻塞队列功能。  总结：   1. 减轻数据库的压力，将经常被访问，但更新不频繁的数据，保存在缓存中，这样可以减少对数据库的操作 2. 代替数据库的存储功能，保存赞和踩，保存手机验证码，保存用户凭证 |

# Redis搭建环境

|  |
| --- |
| $ docker run -v /usr/local/docker/redis/conf/redis.conf:/usr/local/etc/redis/redis.conf -v /usr/local/docker/redis/data:/data --name myredis -p 6379:6379 redis redis-server /usr/local/etc/redis/redis.conf |

# 启动Redis服务

|  |
| --- |
| 启动redis  分为按默认配置启动和使用配置文件来启动（官方建议）    **1,默认方式启动（非守护线程启动）**  直接运行**bin/redis-server**将以前端模式启动  可以通过运行bin/redis-cli客户端操作redis  如下图所示，先开启redis服务，再通过redis-cli客户端去操作redis    启动redis ./redis-server redis.conf  关闭 ./redis-cli shutdown |

# Redis客户端

|  |
| --- |
| 采用redis自带客户端进行操作    指定主机和端口，连接redis服务端  ./redis-cli -h 127.0.0.1 -p 6379  注意：默认不写-h 和 -p则默认为127.0.0.1,6379 |

# 解除本地绑定

|  |
| --- |
| 注意：Redis的低版本默认没有设置仅限本机访问，而高版本有设置，所以需要将高版本的本机绑定注释掉 |

# 安全加固-设置Redis的访问密码

|  |
| --- |
| 修改redis.conf，并重启redis服务  requirepass shizelei |

# Java连接Redis

|  |
| --- |
| 1，搭建Maven工程，引入相关的依赖  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>   1. 使用Jedis提供的API操作Redis     3，密码加密  redis.clients.jedis.exceptions.JedisDataException: NOAUTH Authentication required. |

# 详解5种数据类型

String list hash set zset

# Redis操作-字符串

|  |
| --- |
| 注意：  字符串的值实际不局限于字符串，比如普通的字符串，复杂的字符串如JSON,XML，**数字**，甚至二进制都可以，但不能超过512MB  下面，我们来演示增删改查的基本操作  SET key value  GET key  DEL key  Mset key value [key value...]  Mget key [key...]  Del key [key]  **有用且特殊的命令：自增命令incr**  当存储的字符串是整数时，Redis提供了一个实用的命令INCR，其作用是让当前键值递增，并返回递增后的值。  INCR key  DECR key  INCRBY key increment  DECRBY key decrement    来，Java程序走起！  jedis.mset(**"name"**,**"hgz"**,**"age"**,**"18"**); List<String> mget = jedis.mget(**"name"**, **"age"**);  应用场景：  1，缓存，减少对数据库的访问压力  2，计数系统 点赞 点踩  3，共享session |

# Redis操作-哈希

|  |
| --- |
| KEY----------------VALUE  String-------------（key-value）  在Redis中，哈希类型是指键值本身又是一个键值对结构。  **我们依然围绕增删改查：**  **hSet key filed value**  HSET key field value  HGET key field  HDEL key field [field ...]  H**M**SET key field value [field value ...]  H**M**GET key field [field ...]  **了解跟hash相关的命令：**  计算个数  hlen key  判断field是否存在  hexists key field  获取所有的field-value  hgetall key    Java程序继续走起：  **应用场景：**  内存中需要保存一个对象的信息时，可以更直观保存对象的完整信息  String-->JSON--->{“name”:””,”price”,””} --》对象---》JSON  Hash(操作属性) |

# Redis操作-列表 list

|  |
| --- |
| Redis中的列表类型是用来存储**多个有序**的字符串，**且可重复**，在开发中可以充当栈或队列的角色。  List  栈：先进后出  队列：先进先出  增删改查  LPUSH key value [value ...]，向左边添加元素  LRANGE key start stop，查询列表元素 0 -1 表示查询列表的所有元素  LPOP key，弹出左边的元素  RPUSH key value [value ...]  RPOP key，弹出右边的元素  **如何用上面的命令，分别模拟出队列和栈的效果？**  **队列：先进先出 a b c d a b c d**  **栈：先进后出 a b c d d c b a**    其他命令：  LLEN key，查询列表的长度  LINDEX key index，获取指定索引的值 get(int index)  LSET key index value，设置指定索引的值  Ltrim key start end 保留指定索引范围的值    Java程序走起！！！  List，Stack |

# Redis操作-集合 set

|  |
| --- |
| Redis中的集合也是用来保存多个的字符串元素，不同的是，集合**不允许有重复**的元素，并且集合中的元素是**无序的**。Redis支持对集合取**交集，并集，差集**。  Set  常用命令：set  Sadd key element [element] 添加元素  Scard key 计算元素个数  Smembers key 获取所有的元素  Srem key element [element] 删除元素  Sismember key element 判断元素是否在集合中，存在返回1，否则返回0  Spop key 从集合随机弹出元素    Java程序走起！  **集合间的操作：**  Sinter key [key...] 求交集  Sunion key [key...] 求并集  Sdiff key [key...] 求差集  Sinter**store** newkey key [key...] 保存交集的结果  Sunion**store** newkey key [key...] 保存并集的结果  Sdiff**store** newkey key [key...] 保存差集的结果    **应用场景：**  标签系统 |

# Redis操作-有序集合 zset

|  |
| --- |
| Redis中的有序集合是一种特殊的集合，首先，他**不允许集合中存在重复**元素，其次它是**有序**的，但这个有序的实现方式不同于我们前面讲到的列表，他是基于**分数**来进行排序的。  zset  常用命令：  Zadd key **score** member [**score** member ...] 添加成员，并为每个成员设置分数  Zcard key 计算成员的个数  Zscore key member 计算某个成员的分数  Zrank key member 计算某个成员的排名  Zincrby key increment member 增加成员的分数  Zrange key start end withscores 返回指定排名范围的成员  zrevrange key start end 逆序展示  Zrem key member [member] 删除成员、  应用场景：  排行榜系统 |

# 通过连接池的方式来操作Redis

|  |
| --- |
| **public static** Jedis getJedisByPool(){  *//创建连接池配置对象* JedisPoolConfig config = **new** JedisPoolConfig();  *//最大空闲数* config.setMaxIdle(50);  *//最大连接数* config.setMaxTotal(100);  *//最大等待毫秒数* config.setMaxWaitMillis(20000);   JedisPool pool = **new** JedisPool(config, **"192.168.10.171"**, 6379,2000,**"v5"**);   **return** pool.getResource(); } |

# Spring-data-reids

|  |
| --- |
| Spring-data-redis是spring大家族的一部分，提供了在spring应用中通过简单的配置访问redis服务，对redis底层开发包进行了高度封装，RedisTemplate提供了redis各种操作、异常处理及序列化，支持发布订阅，并对spring 3.1 cache进行了实现。 官网：<http://projects.spring.io/spring-data-redis/> |

# Spring的方式连接Redis

|  |
| --- |
| 1. 引入依赖   Spring为我们提供了一个spring-data-redis来负责实现对redis的整合  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-redis</artifactId>  <version>1.8.8.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.3.8.RELEASE</version> </dependency>  注意，原先的坐标也需要加入进来   1. 配置spring配置文件   连接池，连接工厂，模板对象  <!-- 配置redis连接池对象 -->  <bean id=*"poolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>  <!-- 最大空闲数 -->  <property name=*"maxIdle"* value=*"50"* />  <!-- 最大连接数 -->  <property name=*"maxTotal"* value=*"100"* />  <!-- 最大等待时间 -->  <property name=*"maxWaitMillis"* value=*"20000"* />  </bean>  <!-- 配置redis连接工厂 -->  <bean id=*"connectionFactory"*  class=*"org.springframework.data.redis.connection.jedis.JedisConnectionFactory"*>  <!-- 连接池配置 -->  <property name=*"poolConfig"* ref=*"poolConfig"* />  <!-- 连接主机 -->  <property name=*"hostName"* value=*"192.168.10.169"* />  <!-- 端口 -->  <property name=*"port"* value=*"6379"* />  <!-- 密码 -->  <property name=*"password"* value=*"shizelei"* />  </bean>  <!-- 配置redis模板对象 -->  <bean class=*"org.springframework.data.redis.core.RedisTemplate"*>  <!-- 配置连接工厂 -->  <property name=*"connectionFactory"* ref=*"connectionFactory"* />  </bean>   1. 使用模板对象操作Redis      1. **问题**   **Java客户端可以正常访问存放进去的key，但是如果从服务器的客户端访问却访问不到。**    **原因：**Spring-data-redis提供的模板对象，默认会对我们的信息进行序列化，所以存放进去的key和value都将会有所变化  **通过分析RedisTemplate源码，我们得出默认采用的序列化器是JDK的方式**    **我们可以来改变他的序列化方式：**    重新执行：    同样，内容也被序列化了，如果不想采用这种默认行为  可以将内容的序列化器修也改为String，结果就都只是原先的字符串形式，或者设置为默认的即可 |

# Redis保存自定义对象-序列化

|  |
| --- |
| 注意：如果是保存对象，序列化器不能使用string序列化器，需要使用JDK序列化器 |

# Redis操作-字符串-汇总

|  |
| --- |
| Spring的操作方式：    注意：操作运算，因为value的序列化方式是JDK，所以无法做数值运算    如何解决？如果项目中常用的是对象，那么就需要在程序中单独设置 |

# 结论

|  |
| --- |
| Key，采用字符串序列化器  Value，采用JDK序列化器，这样方便对象的存储  特殊情况，程序里面单独设置即可 |

# Redis操作-哈希

|  |
| --- |
| Spring操作记录 |

# Redis操作-列表

|  |
| --- |
| Spring |

# Redis操作-集合

|  |
| --- |
| 请大家自学api的使用 |

# Redis操作-有序集合

|  |
| --- |
| 请大家自学api的使用 |

# 总结

|  |
| --- |
| 框架跟原生的差异：  第一：他提供了序列化方式，方便我们对对象的存储和获取  第二：方法名跟指令名不完全一致（不建议） |

# 高级篇

|  |
| --- |
|  |

# 事务-基本操作

|  |
| --- |
| MySQL事务的特点：ACID  原子性，持久性，一致性，隔离性  隔离性---》隔离级别  读未提交----》脏读是一个结果  读已提交----》不可重复读  可重复读---》幻读  串行化 ------》不存在幻读  redis的事务是使用multi-exec的命令组合，使用它可以提供两个重要保证：  1、事务是一个被隔离的操作，事务中的方法都会被redis进行序列化并按顺序执行，事务在执行的过程中不会被其他客户端的发出的命令所打断。  2、事务是一个原子性操作，它要么全部执行、要么全部不执行。  multi到exec命令之间的Redis命令将采取进入队列的形式，直至exec命令的出现，才会一次性发送队列的命令去执行。  关注了“刘德华”,user:1:follow 2 user:2:fans 1    Spring  注意：以下这种写法，不能保证在一个事务里面    所以需要这么写 |

# 事务-回滚机制

|  |
| --- |
| 注意：  Redis中，执行格式错误的命令，会导致事务无法提交    如果遇到“命令格式正确”而“数据类型不符合”的情况时，不会进行事务回滚，**这就是Redis事务不严谨的地方**    事务的特性并不像我们之前熟悉的原子性  要么都成功，要么都成功  乐观锁  悲观锁  实现 |

# 事务-乐观锁机制

|  |
| --- |
| 悲观锁，select \* from t\_table for update;采用数据库的锁机制来实现，会锁记录  乐观锁，version，采用版本号递增的方式来维护，不锁记录  Update t\_user set name=?,version=old\_version+1 where id=? and version=old\_version;  name version old\_version  xiaoliu 2 1  ABA问题，采用是某个业务字段来做乐观锁判断的依据，当采用version之后，就不存在该问题  Version  Store  Update t\_product set store=store-1 where id=1 and store=old\_store;--->ABA  CAS（compare And set /compare and swap）  Select old\_version 1  Update old\_version current\_version  Update t\_product set store=new\_store,version=version+1  where id=1 and version=old\_version  更新库存  Store  Select old\_store 100  Update old\_store current\_store  Update t\_product set store=new\_store where id=1 and store=old\_store;  ABA  Store=100  Store=200  Store=100  乐观锁（CAS,version,ABA），悲观锁(for update)    Redis这种方式主要参照了多线程的CAS模式，这种机制，我们称之为乐观锁机制。  演示正常可以提交事务的情况    不能正常提交事务的情况    关于Redis的事务不严谨的问题是依然存在的，这一点在使用的时候要特别注意： |
|  |
|  |
|  |

# 流水线（pipelined）——批处理

|  |
| --- |
| 在现实情况中，redis的读写速度十分快，而系统的瓶颈往往是在网络通信中的延迟。redis可能会在很多时候处于空闲状态而等待命令的到达。为了解决这个问题，可以使用redis的流水线，流水线是一种通讯协议，类似一个队列批量执行一组命令。  **来一段性能的测试做对比：**  1，没有使用流水线技术的写法    2，使用流水线的写法 |

# 发布订阅

|  |
| --- |
| 这个跟我们之前使用消息队列技术一样。  主要有两个关键操作：  消费者：订阅渠道subscribe channel1  生产者：发布消息到渠道 publish channel1 “redis is good”    Spring中如何使用   1. 定义消息监听类，用于接收消息，该类需要实现MessageListener接口   **public** **class** MyMessageListener **implements** MessageListener {  **private** RedisTemplate redisTemplate;  **public** **void** onMessage(Message message, **byte**[] pattern) {  //1.获取渠道名称  String channel = **new** String(pattern);  System.***out***.println("渠道名称:"+channel);  //2.获取消息  **byte**[] body = message.getBody();  String info = **new** String(body);  System.***out***.println("发布的消息:"+info);  //3.采用序列化器转换下  String rightInfo = (String) redisTemplate.getValueSerializer().deserialize(body);  System.***out***.println("发布的消息:"+rightInfo);  }  **public** RedisTemplate getRedisTemplate() {  **return** redisTemplate;  }  **public** **void** setRedisTemplate(RedisTemplate redisTemplate) {  **this**.redisTemplate = redisTemplate;  }  }   1. 在Spring配置文件中，配置该消息监听类，引用RedisTemplate   <!-- 配置消息监听器对象 -->  <bean id=*"messageListener"* class=*"com.javaee.redis.MyMessageListener"*>  <property name=*"redisTemplate"* ref=*"redisTemplate"*/>  </bean>   1. 在Spring配置文件中，配置监听容器   <!-- 配置消息监听容器 -->  <bean id=*"messageListenerContainer"* class=*"org.springframework.data.redis.listener.RedisMessageListenerContainer"*  destroy-method=*"destroy"*>  <!-- redis连接工厂 -->  <property name=*"connectionFactory"* ref=*"connectionFactory"*/>  <!-- 配置监听配置 -->  <property name=*"messageListeners"*>  <map>  <!-- 配置监听器 -->  <entry key-ref=*"messageListener"*>  <!-- 配置监听的渠道 -->  <bean class=*"org.springframework.data.redis.listener.ChannelTopic"*>  <!-- -->  <constructor-arg name=*"name"* value=*"nba"*/>  </bean>  </entry>  </map>  </property>  </bean>   1. 在测试类中发送消息到渠道，观察结果   **redisTemplate**.convertAndSend(**"nba"**,**"Welcome to NBA ALL START"**); |

# 内存管理（重点）

|  |
| --- |
|  |

# 内存管理-超时指令

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1，del的方式，直接删除掉指定的key  2，设置key的超时时间  超时命令：   |  |  | | --- | --- | | 命令 | 作用 | | expire key seconds | 设置超时时间戳，单位为秒 | | ttl key | 查看超时时间，-1表示没有超时时间，-2表示已经超时 | | persist key | 持久化key，取消超时时间 | | expire**at** key timestamp | 设置超时时间点 |     Spring的方式  //超时指令演示  redisTemplate.execute(**new** SessionCallback() {  **public** Object execute(RedisOperations operations) **throws** DataAccessException {  operations.opsForValue().set("k1", "v1");  operations.expire("k1", 60, TimeUnit.***SECONDS***);  Long expire = operations.getExpire("k1", TimeUnit.***SECONDS***);//ttl  System.***out***.println("还有多久过期:"+expire);  operations.persist("k1");  expire = operations.getExpire("k1", TimeUnit.***SECONDS***);  System.***out***.println("还有多久过期:"+expire);  **return** **null**;  }  });  **关键问题：**  如果key超时了，Redis会回收key的存储空间吗？  **答案是不会**。Redis的key超时了，不会被自动回收，**只会标识哪些键值对超时了**。  Redis这么设计的原因是在于避免自动频繁回收，而造成的系统卡顿的情况。  避免出现太多的内存碎片  **那么该如何回收？**   1. 内置的定时回收（这里并不关注怎么回收） redis.conf      1. 惰性回收，使用get命令时，将会触发回收   当执行get命令时，如果该资源已经过时，则进行回收 |

# 内存管理-回收策略（重点）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 算法名称 | 算法描述 | | Volatile-lru | 采用最近使用最少的淘汰策略，Redis将回收那些超时的键值对 | | Allkeys-lru | Redis将对所有的键值对，采用最近最少使用的策略进行淘汰 | | Volatile-random | 采用随机淘汰策略删除超时的键值对 | | Allkeys-random | 采用随机淘汰策略删除所有的键值对 | | Volatile-ttl | 采用删除存活时间最短的键值对策略 | | Noeviction | 不淘汰任何键值对，当内存已满，只支持读，不支持写 |   **注意**：为了提高垃圾回收的策略，LRU和TTL算法都不是精确的算法，而是一个近似的算法。  就是他会根据探测样本的数量设置，来进行探测，然后根据探测到这几个进行淘汰。  A 10 b 20 c 30 d 5  LRU:看命中率  TTL:看时间      总结：   1. 关于key有效期的指令的使用 2. 内部的回收策略   默认不回收  设置回收的频率，默认是10次/秒  设置回收的算法，LRU,random,TTL  算法不是精确的，而是采样的方式，可以设置采样的个数，默认是5个 |

# 持久化机制

|  |
| --- |
| 快照（snapshotting），它是备份当前瞬间Redis在内存中的数据结构  AOF（只追加文件,日志），它的作用是当Redis执行写命令后，在一定的条件下将执行过的写命令依次保存在Redis的文件中。以后依次执行这些保存的命令就可恢复Redis的数据  Set k1 v1  Set k2 v2 |

# 持久化机制-快照(rdb)

|  |
| --- |
| 基于两个点，时间+修改的数量 |

# 持久化机制-AOF

|  |
| --- |
| 1. ----------->80M ----->rewrite----->56M   112M------>rewrite---->  直接将内存中的数据写到文件上，而不是将文件的指令做比较  内存  内存回收：  并不是立马回收，而是有频率回收，  回收算法不是精确（不用全局比较），使用采样的算法  持久化：  Rdb：不是实时持久化，而是按照一定周期和改变的数量级来控制  Aof：策略（每次操作记录一次，每秒记录一次），会在满足一定条件之后，对这个日志文件进行重写（数据恢复的效率提高）  Set k6 v6  Set k6 vv6  Set k6 vvv6  Set k6 vvv6 |