RedisVIP讲义-第一天

RedisVIP目标

不管Redis的已有基础，学完Vip的Redis模块，能够知道Redis的基本使用以及高阶使用（集群、事务、批处理等等）、能够掌握到底层的相关原理源码（数据结构、内存管理、持久化）、解决实战问题、Redis的调优（大key的处理、慢查询）

1. 课程内容设计对标阿里P6+的能力要求
2. 掌握Redis的应用场景及基础入门功能使用
3. 掌握Redis能实现分布式锁等高阶使用。
4. 了解Redis的数据存储结构、内存管理机制的实现原理
5. 掌握Redis的内存管理机制：持久化、过期、淘汰策略
6. 能独立完成Redis集群部署及高可用、以及节点扩容、删减的数据迁移
7. 掌握在Redis缓存雪崩、缓存击穿、缓存穿透、与DB数据一致性等场景问题解决方案

一、RedisVIP安排

我们Redis的安排是6节课 从简单的基本使用、底层的数据结构、高可用、实战、集群、调优等多个方面来讲解Redis相关知识

第一天： Redis基础篇： Redis的由来、特性、基本应用场景

第二天： Redis高阶使用篇：分布式锁、redission分布式锁源码解析；事务、订阅发布

第三天：Redis数据结构篇： Redis总数据结构、不同数据类型Value的底层存储数据结构

第四天：Redis内存管理篇： 内存管理之过期、淘汰机制原理源码解析

第五天：Redis高可用篇：持久化、主从一致性保证、sentinel集群脑裂、cluster集群

第六天：Redis实战篇：Redis数据丢失场景、与DB数据一致性问题、缓存雪崩、击穿、穿透场景分析

二、今日教学目标

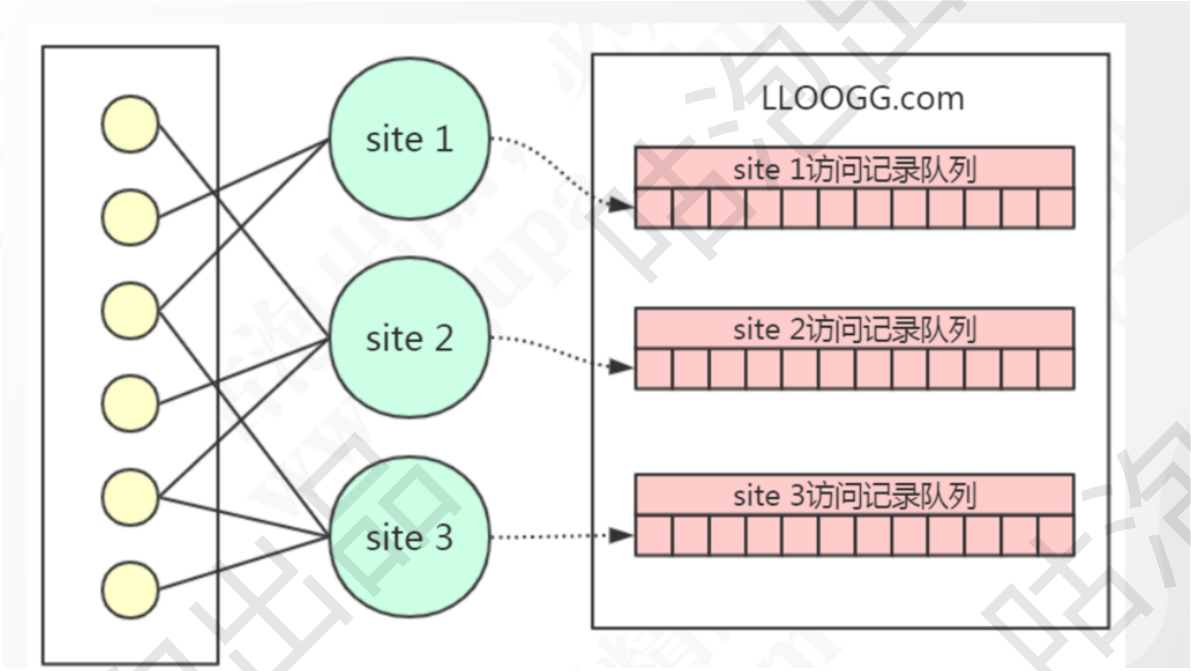
1. 掌握Redis的由来以及核心特性
2. 掌握Redis常用数据类型以及使用场景

三、教学过程（100min）

Redis由来

07年的时候有一个意大利西西里岛的小伙子，笔名antirez（http://invece.org/），创建了一个访客信息网站 LLOOGG.COM。这个网站是干嘛的呢，其实就是跟我们的百度统计一样！需要知道网站的访问情况，比如访客的IP、操作系统、浏览器、使用的搜索关键词、所在地区、访问的网页地址等等。我们不用自己写代码去实现这个功能，只需要在全局的 footer 里面嵌入一段JS 代码就行了，当页面被访问的时候，就会自动把访客的信息发送到这些网站统计的服务器，并且准实时的展示出来。 。

LLOOGG.COM 提供的就是这种功能，它可以查看最多10000条的最新浏览记录。 这样的话，它需要为每一个网站创建一个列表（List），不同网站的访问记录进入到不同的列表。如果列表的长度超过了用户指定的长度，它需要把最早的记录删除（先进先出）。



大家来想一下这样的底层数据怎么来保存了！肯定是用数据库对吧！

他最开始的时候用的就是mysql。但是当 LLOOGG.COM 的用户越来越多的时候，它需要维护的列表数量也越来越多，这种记录最新的请求和删除最早的请求的操作也越来越多。

然后mysql数据还是写在磁盘的，那么每一次记录和删除都要读写磁盘，因为数据量和并发量太大，在这种情况下无论怎么去优化数据库都不管用了。

他考虑到最终限制数据库性能的瓶颈在于磁盘，所以打算放弃磁盘，自己去实现一个具有列表结构的数据库的原型，把数据放在内存而不是磁盘，这样就可以大大地提升列表的操作和查的效率。antirez发现这种思路确实能解决这个问题。

所以他用C语言重写了一个内存数据库！

09年，Redis这个内存数据库出世了。从最开始只支持列表的数据类型，到现在支持多种数据类型，并且提供了一系列的高级特性，Redis已经成为一个在全世界被广泛使用的开源项目。这就是redis的由来！

那么redis的全称是全称是REmote DIctionary Service，直接翻译过来是远程字典服务

官网的大致介绍

所以，我们知道了Redis当初是为了去解决性能问题。而基于内存去操作的一个数据结构存储

官网介绍：https://redis.io/topics/introduction

中文官网：http://www.redis.cn/

Redis特性

快

为什么快?

1. 基于内存操作，操作不需要跟磁盘交互
2. 本身就是k-v结构，类似与hashMap，所以查询速度接近O(1)。
3. 同时redis自己底层数据结构支持，比如跳表、SDS
4. 命令执行是单线程，同时通信采用多路复用
5. IO多路复用，单个线程中通过记录跟踪每一个sock（I/O流） 的状态来管理多个I/O流 。

其他特性:

- 更丰富的数据类型 ， 虽然都是k、v结构， value可以存储很多的数据类型
- 完善的内存管理机制、保证数据一致性：持久化机制、过期策略
- 支持多种编程语言
- 高可用，集群、保证高可用

Redis初识

安装以及Redis文件目录

见安装文档

简单的操作

刚才讲了它是做为一个内存型数据库，肯定是要存数据的。。那怎么存?

我们刚才讲过它是一个key-value的方式

所以存值跟获取值跟我们的hashMap很类似

```
set huihui 666
```

取值

```
get huihui
```

清空实例下的所有数据（慎用,因为是删库跑路的）

```
flushall
```

清空当前数据库

```
flushdb
```

查看所有键(生成环境慎用，因为是单线程的，请求量过多导致线程阻塞，所有耗时的指令都慎用)

```
keys *
```

对某个key设置过期时间

```
expire huihui 10

pexpire huihui 10000
```

查看键是否存在

```
exists huihui
```

删除键

```
del huihui huihui1
```

查看对外类型

```
type huihui2
```

这些是一些比较基本的命令，还有一些命令我就不一一列举了，官网对于指令有完整的完档

官网指令地址：https://redis.io/commands

Redis数据类型以及应用场景

存数据，但是存数据的格式可以有很多 比如我们电脑里面如果要存入一些信息，我们可以有文本、有音频 、有视频等等！那么redis数据也支持不同的数据类型

官网：https://redis.io/docs/manual/data-types/

String、Hash、Set、List、Zset、bitMap（基于String类型）、Hyperloglog（基于String类型）、Geo（地理位置）、Streams流。

今天我们就来讲用得最多的、最常用的几个数据类型的使用以及场景。

Strings

我们刚才已经知道了redis里面都是key-value的结构存储的，那么string数据类型那个value可以存储什么？其实这个value可以存储String,Number,Float,Bits等等.....但是最大的大小是512M。Redis中Key也是基于String类型存储，所以最大大小也是512M

那么这个String到底是怎么玩的，我们来看下string的一些基本指令！

基本指令

批量设置

```
mset huihui niubi huihui1 222
```

批量获取

```
mget huihui huihui1
```

获取长度

```
strlen huihui
```

字符串追求内容

```
append huihui good
```

获取指定范围的字符

```
getrange huihui 0 5
```

(整数) 值递增

```
incr intkey

incrby intkey 100
```

(浮点数) 值递增看

```
set f 2.6

incrbyfloat f 7.3
```

如果不存在这个key设置成功，存在设置失败（分布式锁）

```
set lock1 1 EX 10 NX
```

EX – 设置指定的到期时间(以秒为单位)。

PX - 设置指定的到期时间(以毫秒为单

NX - 仅在键不存在时设置键。

XX - 只有在键已存在时才设置。

```
setnx k1 1
```

String 更多指令参考：https://redis.io/commands/?group=string

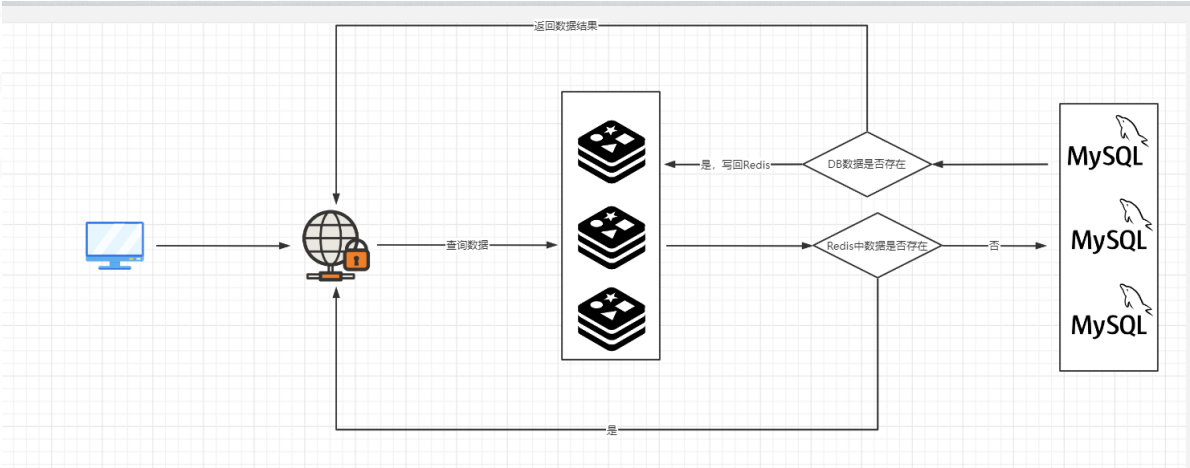
应用场景

缓存

当初 redis出来的目的就是为了快，所以性能肯定比DB快。

那么我们是不是可以把本来应该每次去db查的数据，在第一次查询的时候，把数据放到mysql，这样后续的查找就不用每次都去DB拿，而是优先从redis读取。

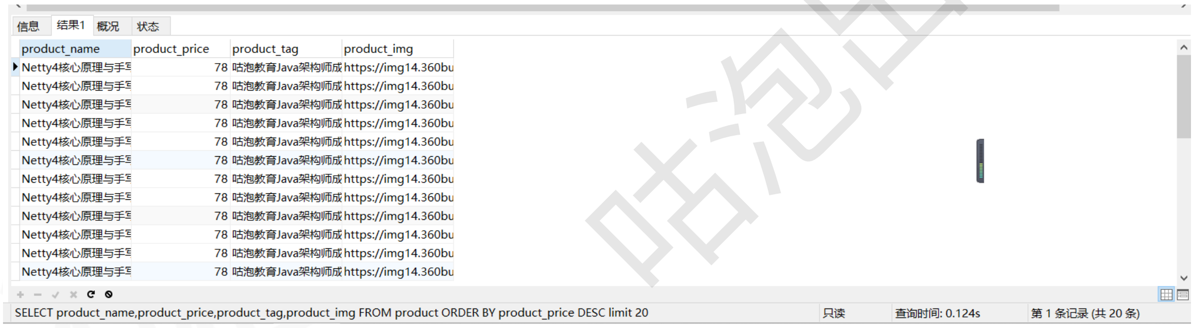
如下图



我们简单来看下这个怎么做？我们现在基于springboot来集成redis 实现一个简单的缓存

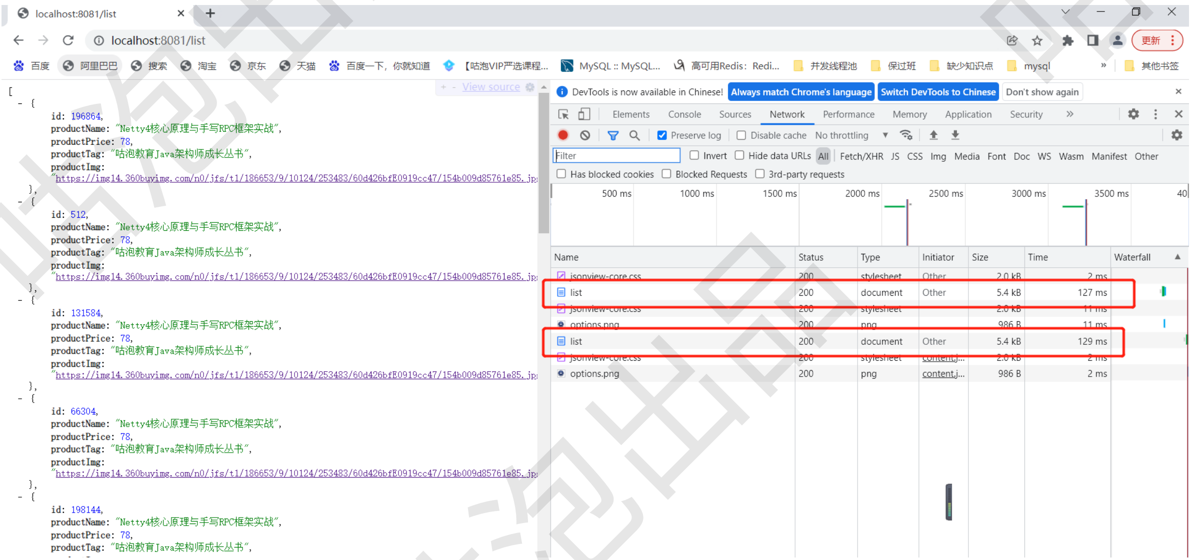
1.假如我们Mysql里面有个表叫做商品表，我要根据价格去排序，获取前20条

```
SELECT product_name,product_price,product_tag,product_img
FROM product ORDER BY product_price DESC limit 20
```



mysql耗时大概在100多ms。

2.通过Mybatis接口查询出来后，接口的返回时间,我们发现在100多ms



3.或许大家觉得OK了，但是如果现在碰到个比较苛刻的老大，或者qps要求很高，我需要优化到10ms。 10ms这个只要去走DB，肯定就达不到，所以，我们可以引用Redis来做缓存。

- pom文件引入redis包

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

- 设置序列化方式

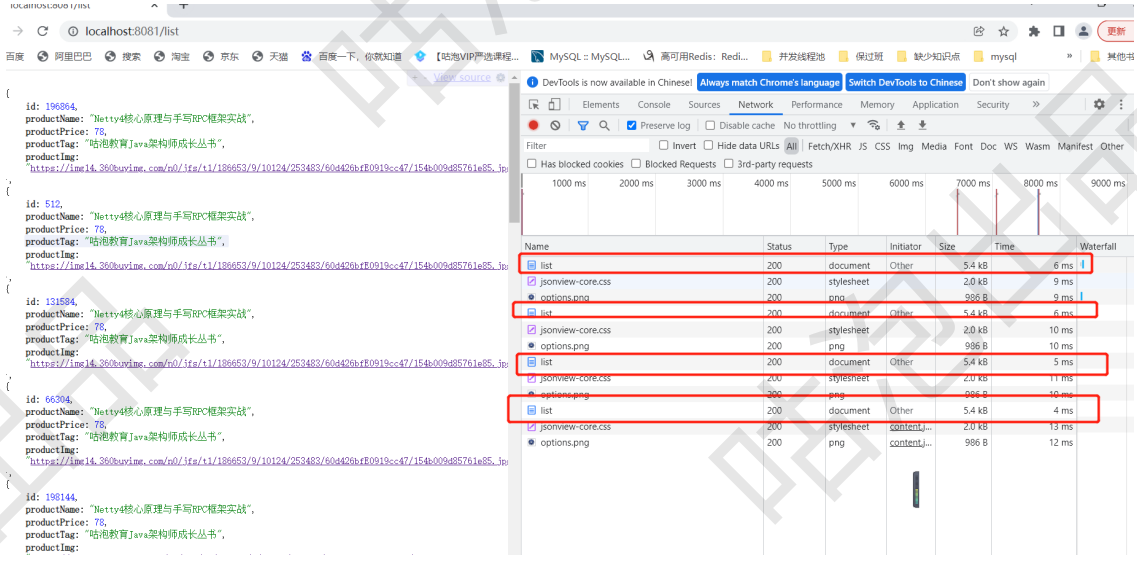
```
@Configuration
@EnableCaching
public class RedisConfig {

    @Bean
    RedisTemplate<String, Object>
redisTemplate(RedisConnectionFactory
redisConnectionFactory) {

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

 redisTemplate.setConnectionFactory(redisConnectionFact
ory);
        Jackson2JsonRedisSerializer
jackson2JsonRedisSerializer = new
Jackson2JsonRedisSerializer(Object.class);
        redisTemplate.setKeySerializer(new
StringRedisSerializer());

 redisTemplate.setValueSerializer(jackson2JsonRedisSeri
alizer);
        redisTemplate.setHashKeySerializer(new
StringRedisSerializer());
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }

}
```

- 更改查询方法，优先去redis查询，没有再去DB查询，再塞回到Redis并且使用dcl解决缓存击穿问题

```
public List<Product> selectProduct() throws
InterruptedException {
    String key = "productList";
    List<Product> products = (List<Product>)
redisTemplate.opsForValue().get(key);
    if (products == null) {
        synchronized (Product.class)  //加锁  为了防止缓存
击穿（后面会讲），防止并发的时候请求全部并发打到DB
```

```
            {
                products = (List<Product>)
redisTemplate.opsForValue().get(key); //双重检查，如果不再
次检查就算加锁，请求还是会打到DB，只不过并行改成了串行
                if (products==null)  //再次判断，保证进入DB只有
一次
                {
                    products =
productMapper.selectProduct();
                    if (products != null) {  //查询数据不为
空，塞回到redis

 redisTemplate.opsForValue().set(key, products, 5,
TimeUnit.MINUTES); //设置时间，因为DB根redis有数据一致性问
题，保证最终一致
                    }
                }
            }
        }
        return products;
}
```

- 查看结果



分布式ID

什么是分布式ID?

我们ID一般用的是数据库自增的ID 每个表的数据的ID都是每个表自增的。所以，假如哪天我要做分表操作，这个时候我们发现有问题，因为ID不同的表是存在可能相同的。

那么如果还简单用数据库自增ID的话，我不同的表里数据不同但是ID相同，我的ID就不能标志数据的一致性。

所以我们解决思路：把生成ID这个步骤独立出来，不要在表中生成，可以交给第三方生成，并且生成的步骤不能有并发问题。

而我们的redis里面刚好有个incr 或者incryby的指令，递增 并且命令执行是单线程的 所以不会有并发导致ID相同。

实例：

```
public  Long getId() {
    return
redisTemplate.opsForValue().increment("productId");
}
```

总结：

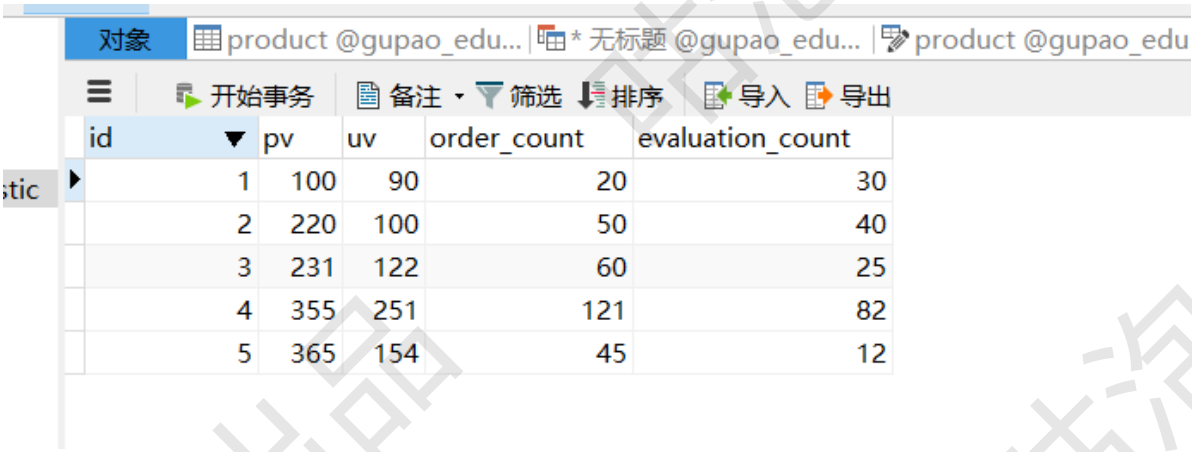
1.缓存相关场景 缓存、 token（跟过期属性完美契合）

2.线程安全的计数场景 （软限流、分布式ID等）

Hashes

我们发现我们string可以存浮点，整型，字符串类型，但是假如我现在接到了一个需求。

刚才我们不是有各种书么，现在假如要去给这些书进行数据统计，统计PV、UV、订单数、评论数等等，我们一般会有个统计表如下



id	pv	uv	order_count	evaluation_count
1	100	90	20	30
2	220	100	50	40
3	231	122	60	25
4	355	251	121	82
5	365	154	45	12

随着商品数量的增多，这个查询效率也会越来越低，那么也得去放到redis缓存。这个时候最合适的不是用String去存储，而是用我们的hash去存储。至于怎么存储，我们先来看下hash的一些简单操作

基本指令

设置与批量设置

```
hset h1 f 6
hset h1 e 5
hmset h1 a 1 b 2 c 3 d 4
hget h1 a
hmget h1 a b c d
hkeys h1
hvals h1
hgetall h1
hincrby h1 a 10   给某个字段添加值
```

Key操作

```
hexists h1 y   查询key中file是否存在

hdel h1  a

hlen h1
```

Hash 更多指令参考：https://redis.io/commands/?group=hash

我们发现是一个大key　filed　value的形式，相比于String类型 多了一个filed。

使用场景

String可以做的事情，Hash都可以做！为什么这么说，因为hash就比String多了一层key而已！并且单个filed可以单独更改

1. 存储对象类的数据(官网说的)　比如一个对象下有多个字段
2. 统计类的数据　我可以对单个统计数据进行单独操作
3. 购物车（一般人不会这样做，存在数据丢失（后面会讲））

   我们先来看下购物车应该要实现哪些功能

   1. 添加商品数量，我买了1个，再想来个
   2. 贼有钱，全选付款
   3. 没钱了，移除商品！

   好。大家根据这3个想下，我们应该会用到哪些指令！

Demo

Redis的客户端很多很多：https://redis.io/docs/clients/

由于在明天我们高阶使用会讲到Redisson的客户端,所以今天我们的Demo基于Redssion来做，但是我们Springboot是没有集成Redission的，所以我们需要引入Redission包

```
<dependency>
    <groupId>org.redisson</groupId>
    <artifactId>redisson</artifactId>
    <version>3.17.1</version>
</dependency>
```

创建连接

```
public class RedissonClientUtil {
    public static RedissonClient getClient() {
        Config config = new Config();

 config.useSingleServer().setAddress("redis://192.168.8.12
8:6379");
        config.setCodec(new StringCodec());
        return Redisson.create(config);
    }
}
```

Redission主要优势：不是基于Redis的API开发，而是基于我们的业务做了各类封装，底层基于netty实现异步。这个在我们等下的demo中都会有体现。

```
public static void main(String[] args) {
        RedissonClient redissonClient =
RedissonClientUtil.getClient();
        //先返回不同需求的对象 RMap，实现类都会继承
RedissonObject
        RMap<String, Integer> productStatics =
redissonClient.getMap("productStatics:1");
//        productStatics.fastReplaceAsync("pv",1);
        Map<String,Integer> map=new HashMap<>();
        map.put("pv",1);
        map.put("uv",1);
        productStatics.putAll(map);

 System.out.println(productStatics.addAndGet("pv",1));

    }
```

Lists

存储有序的字符串列表，元素可以重复。列表的最大长度为 2^32 - 1 个元素（4294967295，每个列表超过 40 亿个元素）。

基本指令

左右添加元素

```
lpush queue a

lpush queue b c
```

右边添加元素

```
Rpush queue d e
```

左右弹出第一条

```
lpop queue

rpop queue
```

那么在弹出命令前面加个b，又是个什么操作，这个就是弹出的时候如果发现没有可以弹出的

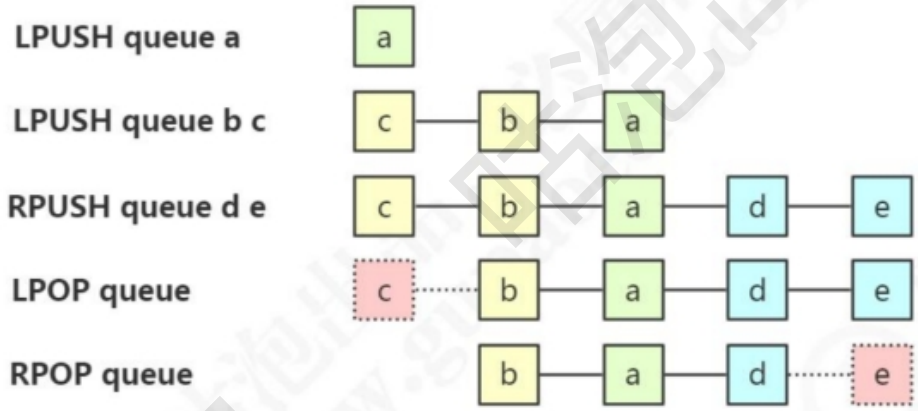
左右弹出第一条，并设置超时，直到有数据弹出或者超时

```
blpop queue 10

brpop queue 10
```

既然是list，我们也要知道怎么取值

取值

```
lindex queue 0

lrange queue 0 -1
```



List 更多指令参考：https://redis.io/commands/?group=list

应用场景

1. 用户消息时间线

因为list是有序的 所以我们可以先进先出 先进后出的列表都可以做

```
      RedissonClient redissonClient =
RedissonClientUtil.getClient();
        RDeque<String> msgList =
redissonClient.getDeque("msgList");
        msgList.addFirst("周一:我和床才是原配，工作只是第三者");
        msgList.addFirst("周二:苍天啊！大地啊！时光时光快些
吧！");
        msgList.addFirst("周三:就这样吧，我已悟透");
        msgList.addFirst("周四:明天周五了~");
        msgList.addFirst("周五:这周末我要先这样这样再那样那样，舒
舒服服啊！");
        for (int i = 0; i <5 ; i++) {
            System.out.println(msgList.poll());
        }
```

2. 消息队列

List提供了两个阻塞的弹出操作：BLPOP/BRPOP，可以设置超时时间。

BLPOP：BLPOP key1 timeout 移出并获取列表的第一个元素， 如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止。

```
RBlockingDeque<String> blockingDeque =
redissonClient.getBlockingDeque("blockList");  //阻塞获取
while (true) {
  System.out.println(blockingDeque.poll(1000,
TimeUnit.SECONDS));
}
```

其他客户端往blockList添加数据，能够阻塞获取

```
127.0.0.1:6379> lpush blockList 8
```

但是队列我不建议大家用redis了，为什么，因为我们有更加成熟的技术，就是我们的MQ！

并且redis它是内存型数据库，可能会造成数据丢失，还有它消费了后没法回应！没有ack机制

Sets

String类型的无序集合，最大存储数量 2^32-1（40 亿左右）。 并且他们的添加、删除元素的时间都是O（1）。

基本指令

添加一个或者多个元素

```
sadd huihuiset a b c d e f g
```

获取所有元素

```
smembers huihuiset
```

获取所有元素的个数

```
scard huihuiset
```

随机获取一个元素

```
srandmember huihuiset
```

随机弹出一个元素

```
spop huihuiset
```

弹出指定元素

```
srem huihuiset g a
```

查看元素是否存在

```
sismember huihuiset e
```

获取前一个集合有而后面1个集合没有的

```
sdiff huihuiset huihuiset1
```

获取交集

```
sinter huihuiset huihuiset1
```

获取并集

```
sunion huihuiset huihuiset1
```

Sets 更多指令参考：https://redis.io/commands/?group=set

应用场景

1. 抽奖 spop跟srandmember 随机弹出或者获取元素

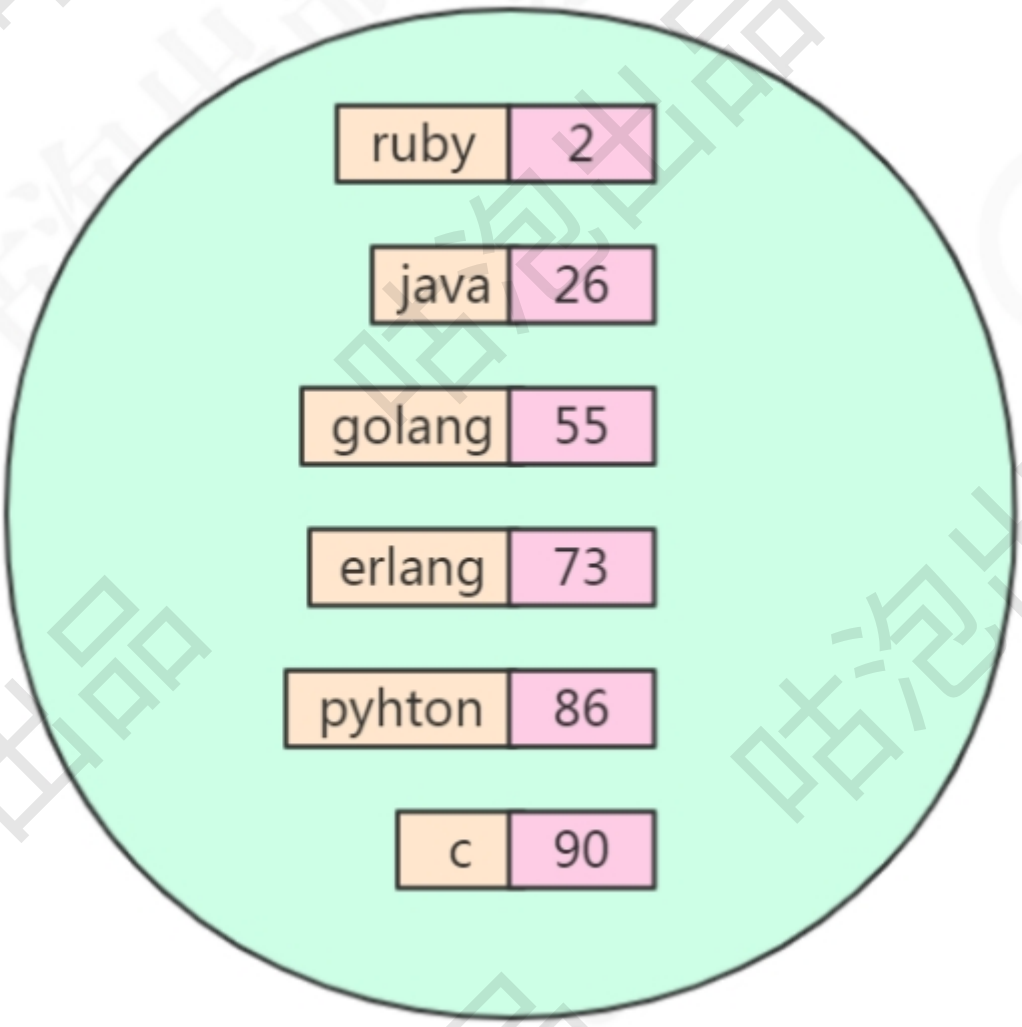
```
    RSet<String> gupao =
redissonClient.getSet("gupao");
    List<String> members= Arrays.asList("灰
灰","james","mic","tom","颖子","梦情","田田","云帆","文
文","荣荣",
                                        "依依","丸子","橙
子","君君","蜀婷","书韵");
    gupao.addAll(members);
    //随机抽取三等奖3个
    RFuture<Set<String>> setRFutureThree =
gupao.removeRandomAsync(3);
    //随机抽取二等奖2个
    RFuture<Set<String>> setRFutureSecond =
gupao.removeRandomAsync(2);
    //随机抽取一等奖1个
    RFuture<Set<String>> setRFutureOne =
gupao.removeRandomAsync(1);
    System.out.println("三等
奖："+setRFutureThree.get());
    System.out.println("二等
奖："+setRFutureSecond.get());
    System.out.println("一等奖："+setRFutureOne.get());
```

2. 点赞、签到等 sadd 集合存储
3. 交集并集 关注等场景

```
    RedissonClient redissonClient =
RedissonClientUtil.getClient();
    RSet<String> huihui = redissonClient.getSet("灰灰");
    //huihui关注的人
    List<String> huihuiMembers=
Arrays.asList("james","mic","tom","颖子","梦情");
    huihui.addAll(huihuiMembers);

    RSet<String> james =
redissonClient.getSet("james");
    //james关注的人
    List<String> jamesMembers= Arrays.asList("颖
子","mic","tom","云帆","梦情","荣荣",
```

```
                                                    "依依","丸
子","橙子","君君","蜀婷","书韵");
        james.addAll(jamesMembers);
        //共同关注的人
        RFuture<Set<String>> interSet =
huihui.readIntersectionAsync("james");
        System.out.println("共同关注的人："+interSet.get());

        //灰灰可能认识的人
        RFuture<Set<String>> difSet =
james.readDiffAsync("灰灰");
        System.out.println("灰灰可能认识的人："+difSet.get());
```

Sorted Set(ZSet)



sorted set，有序的set，每个元素有个score。 我们看张图!

score相同时，按照key的ASCII码排序。

基本指令

批量添加

```
zadd z1 10 a 20 b 30 d 40 c
```

根据分数从低到高

```
zrange z1 0 -1 withscores
```

根据分数从高到低

```
zrevrange z1 0 -1 withscores
```

根据分数范围获取值

```
zrangebyscore z1 20 30
```

移除元素

```
zrem z1 a
```

获取zset个数

```
zcard z1
```

给某个元素加分

```
zincrby z1  20 b
```

获取范围内的个数

```
zcount z1 50 60
```

返回指定元素的索引

```
zrank z1  d
```

获取元素的分数

```
zscore z1 d
```

Sets 更多指令参考：https://redis.io/commands/?group=sorted_set

应用场景

1. 排行榜

```
    RedissonClient redissonClient =
RedissonClientUtil.getClient();
    // 现在灰灰公司要做绩效考核 ，默认都是50分
    RScoredSortedSet<String> huihuiCompany =
redissonClient.getScoredSortedSet("huihuiCompany");
    huihuiCompany.add(50, "lily");
    huihuiCompany.add(50, "lucy");
    huihuiCompany.add(50, "zhangsan");
    huihuiCompany.add(50, "lisi");
    huihuiCompany.add(50, "wangwu");
    huihuiCompany.add(50, "liuliu");

    //lisi给灰灰老师买了杯咖啡，加50分
    huihuiCompany.addScore("lisi", 50);
    //lily长得不错，60分
    huihuiCompany.addScore("lily", 50);
    //wangwu 不听话，减10分
    huihuiCompany.addScore("wangwu", -10);
    //liuliu说灰灰老师坏话，扣20
    huihuiCompany.addScore("liuliu", -20);
    //张三拍灰灰老师马屁，加10分
    huihuiCompany.addScore("zhangsan", 10);
    //lucy不上班，开除
    huihuiCompany.remove("lucy");
    RFuture<Collection<ScoredEntry<String>>>
collectionRFuture =
huihuiCompany.entryRangeReversedAsync(0, -1);
    Iterator<ScoredEntry<String>> iterator =
collectionRFuture.get().iterator();
    System.out.println("绩效从高到低：");
```

```
    while (iterator.hasNext()) {
        ScoredEntry<String> next = iterator.next();
        System.out.println(next.getValue());
    }

    RFuture<Collection<ScoredEntry<String>>>
collectionRFuture1 =
huihuiCompany.entryRangeReversedAsync(0, 2);
    Iterator<ScoredEntry<String>> iterator1 =
collectionRFuture1.get().iterator();
    System.out.println("绩效前三名：");
    while (iterator1.hasNext()) {
        ScoredEntry<String> next = iterator1.next();
        System.out.println(next.getValue());
    }
```

这个排行榜一般在公司的使用是在任何排行的地方都可以用到，像销售榜、热搜榜、游戏评分排行等等

BitMap

位图不是实际的数据类型，而是String类型中定义的一种面向位的操作，所以这个位图的最大长度是512M。

可以容纳最少2^32 不同的位。可以在不同的位置设置0或者1

基本指令

设置位的值

```
setbit permission 5 1  //把位5设置为1
```

获取位的值

```
getbit permission 5   //得到位5的值
```

获取key的为1的个数

```
bitcount permission    //获取位为1的总数
```

获取0或者1的第一位

```
bitpos permission 1    //获取 permission位为1的第一个位置
```

获取多个bitmap的位操作，比如& |

```
bitop AND hbit bitkey permission  //获取bitkey与permission的&运算 并且赋值给hbit
```

应用场景

1.实时的统计数据

比如：学员到课率 假如学生huihui

每次来上课就将相关的位记录为1：

假如第一天、第五天、第10天到了,每次来了设置为1

```
setbit huihuibit 1 1  //把位5设置为1
setbit huihuibit 5 1  //把位5设置为1
setbit huihuibit 10 1  //把位5设置为1
```

我们可以统计这个用户来了几天

```
bitcount huihuibit    //得到总共到课几天
```

我们也可以统计一个网站一天有多少用户访问：

假如gupaobit：2022-06-28网站 有ID 为155 188 199 这3个用户访问

```
127.0.0.1:6379> setbit gupao:2022-06-28 155 1
(integer) 0
127.0.0.1:6379> setbit gupao:2022-06-28 188 1
(integer) 0
127.0.0.1:6379> setbit gupao:2022-06-28 199 1
(integer) 0
127.0.0.1:6379> bitcount gupao:2022-06-28
(integer) 3
127.0.0.1:6379>
```

我们可以得到6.28这天有3个用户访问我们的网站

2.存储与ID相关的节省空间并且高性能的 是或者否的值

用户权限 ： 可以用不同的位来代表不同的权限 如果有权限设置为1 否则设置为0 这样我们就能很高效的拿到用户是否有相关权限。

四、课后总结

1．我们知道了redis的由来，它是因为意大利西西里岛的一个聪明的小伙子，在做一个访客信息网站时候，为了解决在用mysql的时候带来的性能压力，发明的一个内存型数据库

2．所以就有了它的特性，就是要快

1. 基于内存操作，操作不需要跟磁盘交互
2. 本身就是k-v结构，类似与hashMap，所以查询速度接近O(1)。
3. 同时redis自己底层数据结构支持，比如跳表、SDS
4. 命令执行是单线程，同时通信采用多路复用
5. IO多路复用，单个线程中通过记录跟踪每一个sock（I/O流） 的状态来管理多个I/O流 。

其他特性：

- 更丰富的数据类型 ， 虽然都是k、v结构， value可以存储很多的数据类型

- 完善的内存管理机制、保证数据一致性：持久化机制、过期策略
- 支持多种编程语言
- 高可用，集群、保证高可用
-

我们学习了redis常用的5种数据类型以及使用场景！

（5）String 缓存（性能提升、token存储），计数场景（软限流）等等

（6）Hash 有子键filed 购物车等场景

（7）List 有序集合，元素可以重复 队列，有序列表 阻塞队列

（8）Set 无序集合，元素不可重复 关注点赞，抽奖 、集合类相关操作

（9）ZSet 有序集合，元素不可重复 排行榜

五、下节课预告（5min）

1.Redis高阶使用之分布式锁 与Redisson分布式源码

需要提前掌握：

1. 了解什么是分布式
2. 并发相关知识（原子性问题、串行、并行概念）、多线程相关知识（线程的操作）
3. 掌握锁相关概念，最好看过lock源码