

# 极客时间 Java 进阶训练营

## 第 22 课

### 分布式缓存-Redis 详解



KimmKing

Apache Dubbo/ShardingSphere PMC

# 个人介绍

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

# 目录

1. Redis 基本功能
2. Redis 六大使用场景
3. Redis 的 Java 客户端
4. Redis 与 Spring 整合
5. Redis 高级功能
6. 总结回顾与作业实践

# 1. Redis 基本功能

# Redis 安装

## 三种方式：

- 下载安装、编译源码安装（Windows: 微软提供3.x/Memurai 提供5.x）
- brew、apt、yum 安装
- docker 方式启动

```
docker pull redis
```

```
docker run -itd --name redis-test -p 6379:6379 redis
```

```
docker image inspect redis:latest|grep -i version
```

```
docker exec -it redis-test /bin/bash
```

```
$ redis-cli
```

```
> info
```

# Redis 安装-Docker

Docker 安装与使用演示

新入门注意的坑：没有 redis.conf 文件

如何处理？

```
$ docker run -p 6379:6379 --name redis01 -v /etc/redis/redis.conf:/etc/redis/redis.conf  
-v /etc/redis/data:/data -d redis redis-server /etc/redis/redis.conf --appendonly yes
```

推荐使用 docker 方式

# Redis 性能测试

可以使用自带的命令 redis-benchmark

在我启动的 docker 里 (4Core, 2G)

```
# redis-benchmark -n 100000 -c 32 -t SET,GET,INCR,HSET,LPU SH,MSET -q
```

输出结果:

SET: 82372.32 requests per second

GET: 93109.87 requests per second

INCR: 98328.42 requests per second

LPU SH: 87489.06 requests per second

HSET: 77279.75 requests per second

MSET (10 keys): 70175.44 requests per second

更详细的可以参考: redis-benchmark -n 100000 -c 32

# Redis 的5种基本数据结构

- 1.字符串（string）~ 简单来说就是三种：int、string、byte[]

字符串类型是 Redis 中最为基础的数据存储类型，它在 Redis 中是二进制安全的，这便意味着该类型可以接受任何格式的数据，如 JPEG 图像数据或 json 对象描述信息等。在 Redis 中字符串类型的 value 最多可以容纳的数据长度是512M。

set/get/getset/del/exists/append

incr/decr/incrby/decrby

注意：

- 1、字符串 append：会使用更多的内存
- 2、整数共享：如何能使用整数，就尽量使用整数，限制了 redis 内存+LRU
- 3、整数精度问题：redis 大概能保证16~，，17-18位的大整数就会丢失精确



# Redis 的5种基本数据结构

## - 2.散列 (hash) - Map ~ Pojo Class

Redis 中的 Hash 类型可以看成具有 String key 和 String value 的 map 容器。所以该类型非常适合于存储对象的信息。如 Username、password 和 age。如果 Hash 中包含少量的字段，那么该类型的数据也将仅占用很少的磁盘空间。

hset/hget/hmset/hmget/hgetall/hdel/hincrby

hexists/hlen/hkeys/hvals

=> hashmap 的方法

# Redis 的5种基本数据结构

## - 3.列表（list）~ java 的 LinkedList

在 Redis 中，List 类型是按照插入顺序排序的字符串链表。和数据结构中的普通链表一样，我们可以在其头部（Left）和尾部（Right）添加新的元素。在插入时，如果该键并不存在，Redis 将为该键创建一个新的链表。与此相反，如果链表中所有的元素均被移除，那么该键也将会被从数据库中删除。

lpush/rpush/lrange/lpop/rpop

# Redis 的5种基本数据结构

- 4.集合 (set) ~ java 的 set, 不重复的 list

在 redis 中, 可以将 Set 类型看作是没有排序的字符集合, 和 List 类型一样, 我们也可以在该类型的数值上执行添加、删除和判断某一元素是否存在等操作。这些操作的时间复杂度为 $O(1)$ , 即常量时间内完成依次操作。

和List类型不同的是, Set 集合中不允许出现重复的元素。

sadd/srem/smembers/sismember ~ set.add, remove, contains,  
sdiff/sinter/sunion ~ 集合求差集, 求交集, 求并集

# Redis 的5种基本数据结构

## - 5.有序集合 (sorted set)

sortedset 和 set 极为相似，他们都是字符串的集合，都不允许重复的成员出现在一个 set 中。他们之间的主要差别是 sortedset 中每一个成员都会有一个分数与之关联。redis 正是通过分数来为集合的成员进行从小到大的排序。sortedset 中分数是可以重复的。

`zadd key score member score2 member2...`：将成员以及该成员的分数存放到 sortedset 中

`zscore key member`：返回指定成员的分数

`zcard key`：获取集合中成员数量

`zrem key member [member...]`：移除集合中指定的成员，可以指定多个成员

`zrange key start end [withscores]`：获取集合中脚注为 start-end 的成员，[withscores]参数表明返回的成员包含其分数

`zrevrange key start stop [withscores]`：按照分数从大到小的顺序返回索引从 start 到 stop 之间的所有元素（包含两端的元素）

`zremrangebyrank key start stop`：按照排名范围删除元素

# Redis 的3种高级数据结构

- Bitmaps: setbit/getbit/bitop/bitcount/bitpos

bitmaps 不是一个真实的数据结构。而是 String 类型上的一组面向 bit 操作的集合。由于strings 是二进制安全的 blob，并且它们的最大长度是512m，所以bitmaps能最大设置 $2^{32}$ 个不同的 bit。

- Hyperloglogs: pfadd/pfcount/pfmerge

在 redis 的实现中，您使用标准错误小于1%的估计度量结束。这个算法的神奇在于不再需要与需要统计的项相对应的内存，取而代之，使用的内存一直恒定不变。最坏的情况下只需要12k，就可以计算接近 $2^{64}$ 个不同元素的基数。

- GEO: geoadd/geohash/geopos/geodist/georadius/georadiusbymember

Redis 的 GEO 特性在 Redis3.2版本中推出，这个功能可以将用户给定的地理位置（经度和纬度）信息储存起来，并对这些信息进行操作。

# Redis 到底是单线程，还是多线程？

这个问题本身就是个坑

## IO线程：

- redis 6之前（2020年5月），单线程
- redis 6之后，多线程，NIO 模型 ==> 主要的性能提升点

## 内存处理线程：

- 单线程 ==> 高性能的核心

## 2. Redis 六大使用场景

# Redis 使用场景-1.业务数据缓存 \*

经典用法：

- 1、通用数据缓存，string，int，list，map 等。
- 2、实时热数据，最新500条数据。
- 3、会话缓存，token 缓存等。



# Redis 使用场景-2.业务数据处理

- 1、非严格一致性要求的数据：评论，点击等。
- 2、业务数据去重：订单处理的幂等校验等。
- 3、业务数据排序：排名，排行榜等。

# Redis 使用场景-3.全局一致计数 \*

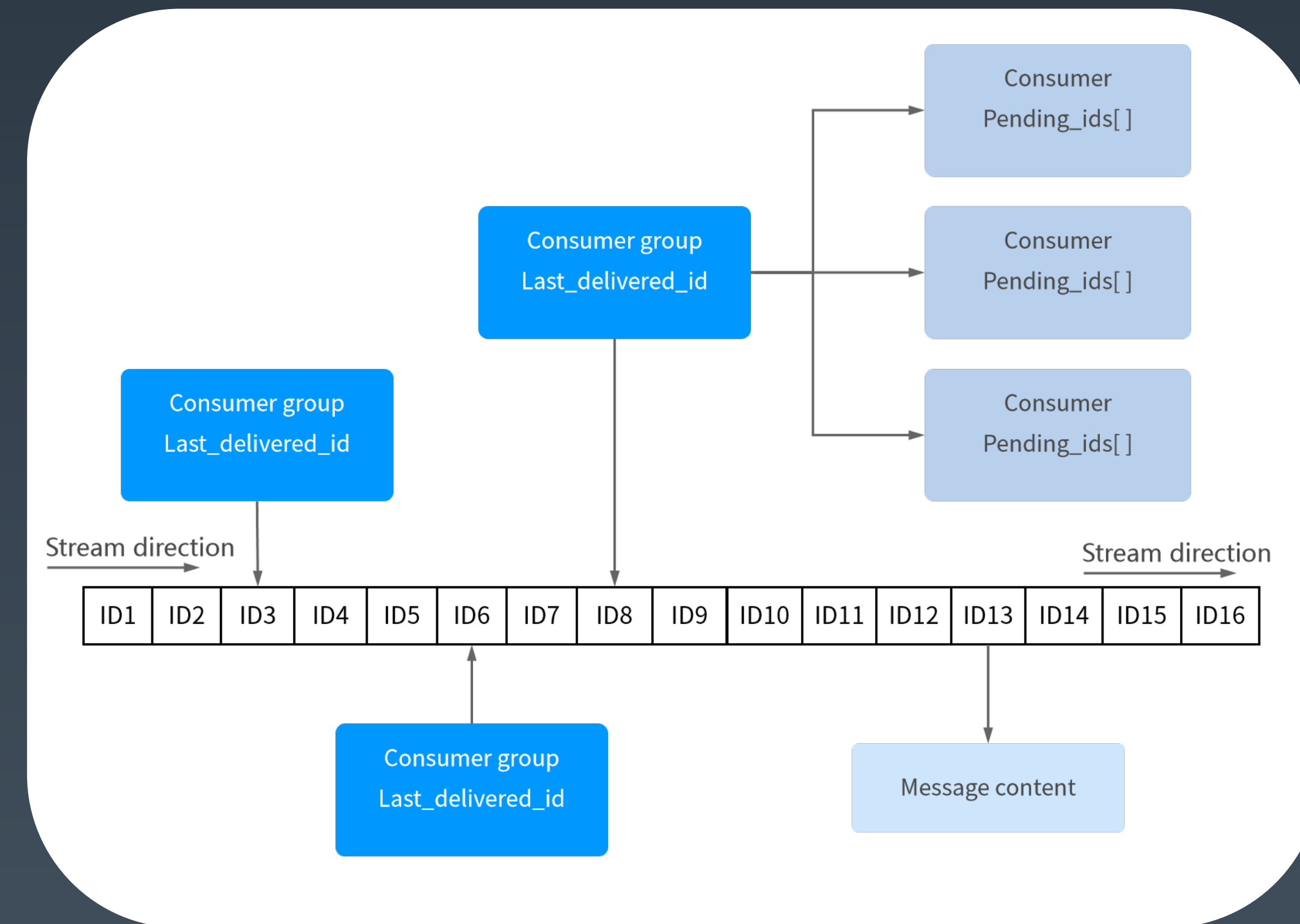
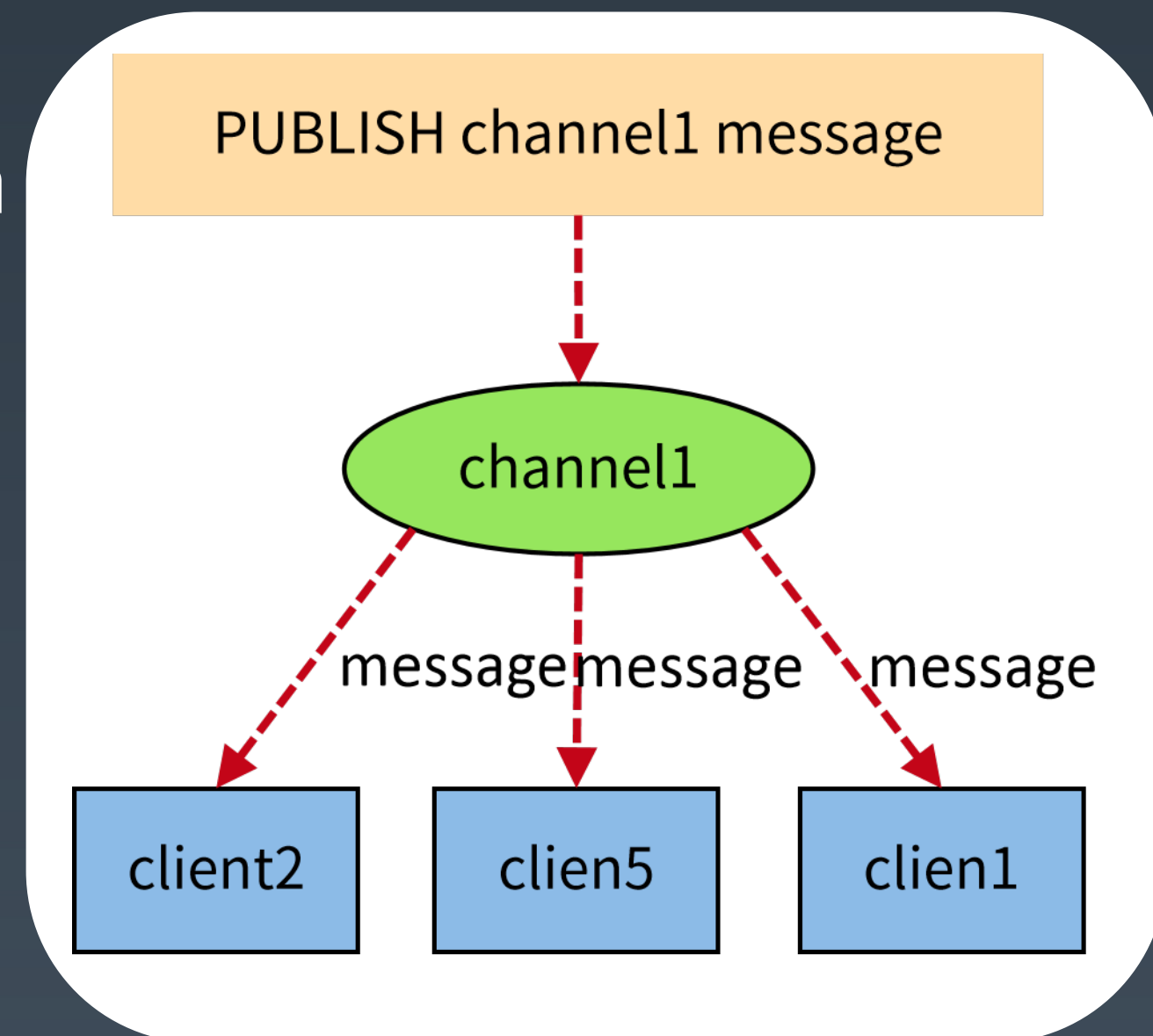
- 1、全局流控计数
- 2、秒杀的库存计算
- 3、抢红包
- 4、全局 ID 生成

# Redis 使用场景-4.高效统计计数

- 1、id 去重，记录访问 ip 等全局 bitmap 操作
- 2、UV、PV 等访问量==>非严格一致性要求

# Redis 使用场景-5.发布订阅与 Stream

1、Pub-Sub 模拟队列  
subscribe comments  
publish comments java



2、Redis Stream 是 Redis 5.0 版本新增加的数据结构。  
Redis Stream 主要用于消息队列（MQ，Message Queue）。  
具体可以参考 <https://www.runoob.com/redis/redis-stream.html>

# Redis 使用场景-6.分布式锁 \*

## 1、获取锁--单个原子性操作

```
SET dlock my_random_value NX PX 30000
```

## 2、释放锁--lua脚本-保证原子性+单线程，从而具有事务性

```
if redis.call("get",KEYS[1]) == ARGV[1] then
    return redis.call("del",KEYS[1])
else
    return 0
end
```

**关键点：**原子性、互斥、超时

## 3. Redis 的 Java 客户端

# Redis 的 Java 客户端-Jedis

官方客户端，类似于 JDBC，可以看做是对 redis 命令的包装。

基于 BIO，线程不安全，需要配置连接池管理连接。

# Redis 的 Java 客户端-Lettuce

目前主流推荐的驱动，基于 Netty NIO，API 线程安全。



# Redis 的 Java 客户端-Redisson

基于 Netty NIO, API 线程安全。

亮点：大量丰富的分布式功能特性，比如 JUC 的线程安全集合和工具的分布式版本，分布式的基本数据类型和锁等。

## 4. Redis 与 Spring 整合

# Spring Data Redis

核心是 RedisTemplate (可以配置基于 Jedis, Lettuce, Redisson))

使用方式类似于 MongoDBTemplate, JdbcTemplate 或 JPA

封装了基本 redis 命令操作:

```
son2JsonRedisSerializer);  
ckson2JsonRedisSerializer); redisTemplate.b|  
m boundGeoOps(Object key) BoundGeo  
m boundHashOps(Object key) BoundHas  
m boundListOps(Object key) BoundLis  
m boundValueOps(Object ke... BoundValu  
m boundSetOps(Object key) BoundSe  
m boundZSetOps(Object key) BoundZSe  
m setBeanClassLoader(ClassLoader classl  
m isEnabledDefaultSerializer()
```

# Spring Boot 与 Redis 集成

引入 spring-boot-starter-data-redis

配置 spring redis

# Spring Cache 与 Redis 集成

默认使用全局的 CacheManager 自动集成

使用 ConcurrentHashMap 或 ehcache 时，不需要考虑序列化问题。

redis 的话，需要：

- 1、默认使用 java 的对象序列化，对象需要实现 Serializable
- 2、自定义配置，可以修改为其他序列化方式

# MyBatis 项目集成 cache 示例

- 1、集成 spring boot 与 mybatis，实现简单单表操作，配置成 rest 接口
- 2、配置 ehcache+mybatis 集成，实现 mybatis 二级缓存
- 3、配置 spring cache+ehcache 缓存，实现方法级别缓存
- 4、修改 spring cache 使用 redis 远程缓存代替 ehcache 本地缓存
- 5、修改 spring cache 使用 jackson json 序列化代替 java 序列化
- 6、整个过程中，使用 wrk 压测 rest 接口性能，并分析为什么？
- 7、尝试调整各种不同的配置和参数，理解 cache 原理和用法。

后面2步留给大家作为作业

## 5. Redis 高级功能

# 1、Redis 事务

## - Redis 事务命令：

开启事务：multi

命令入队

执行事务：exec

撤销事务：discard

## - Watch 实现乐观锁

watch 一个 key，发生变化则事务失败



## 2、Redis Lua ~ open resty = nginx + lua jit

- 类似于数据库的存储过程, mongodb 的 js 脚本

### > 直接执行

```
eval "return'hello java'" 0
```

```
eval "redis.call('set',KEYS[1],ARGV[1])" 1 lua-key lua-value
```

### > 预编译

```
script load script 脚本片段
```

返回一个 SHA-1 签名 shastring

```
evalsha shastring keynum [key1 key2 key3 ...] [param1 param2 param3 ...]
```

1、单线程 -> 脚本执行具有原子性, 操作不会被打断, 值不会被修改

2、每个脚本执行, 都能保证事务

## 3、Redis 管道技术 (pipeline)

合并操作批量处理，且不阻塞前序命令：

```
% (echo -en "PING\r\n SET pkey redis\r\nGET pkey\r\nINCR visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 1) | nc localhost 6379
```

输出：

+PONG

+OK

\$5

redis

:1

:2

:3

对比 redis 的批量操作命令，这里的命令之间没有任何关系

# Redis 数据备份与恢复--RDB ~ frm

## 备份

执行 save 即可在 redis 数据目录生成数据文件 dump.rdb

也可以异步执行 bgsave

## 恢复

将备份文件 (dump.rdb) 移动到 redis 数据目录并启动服务即可

查看文件夹 CONFIG GET dir

```
127.0.0.1:6379> CONFIG GET dir
```

1) "dir"

2) "/data"

# Redis 数据备份与恢复--AOF ~ binlog

## 备份

如果 appendonly 配置为 yes，则以 AOF 方式备份 Redis 数据，那么此时 Redis 会按照配置，在特定的时候执行追加命令，用以备份数据。

```
appendfilename "appendonly.aof"
```

```
# appendfsync always
```

```
# appendfsync everysec
```

```
# appendfsync no.....
```

AOF 文件和 Redis 命令是同步频率的，假设配置为 always，其含义为当 Redis 执行命令的时候，则同时同步到 AOF 文件，这样会使得 Redis 同步刷新 AOF 文件，造成缓慢。而采用 everysec 则代表每秒同步一次命令到 AOF 文件。

## 恢复

自动加载

# Redis 性能优化

## - 内部的核心优化点:

### 1、内存优化 ~ 10G/20G

<https://redis.io/topics/memory-optimization>

hash-max-ziplist-value 64

zset-max-ziplist-value 64

### 2、CPU 优化 ~ 单线程，要命！！！！

1) 不要阻塞，特别是 lua 脚本

2) 谨慎使用范围操作

3) SLOWLOG get 10 //默认10毫秒，默认只保留最后的128条

# Redis 分区 ~ 数据库垂直拆分

设计规划问题：

1、容量 ~ 多个业务系统，共用一个redis，还是应该分开？

规划好key，特别是前缀，a.b.c.d

2、分区

如果缓存数据变大了，就可以分区，order

业务研发团队，一般看不到 redis/tair 实际配置的，怎么用？

1、申请缓存资源，key/token, mall.gouwuche.detail,

2、cache, key=mall.gouwuche.detail

# Redis 使用的一些经验

## 1、性能：

- 1) client~线程数(4~8)与连接数(redis ~ 10000);
- 2) 监控系统读写比和缓存命中率(N:1, 90%+);

## 2、容量：

- 1) 做好容量评估，合理使用缓存资源；监控要注重增量变化。

## 3、资源管理和分配：

- 1) 尽量每个业务集群单独使用自己的 Redis，不混用；
- 2) 控制 Redis 资源的申请与使用，规范环境和 Key 的管理（以一线互联网为例）；
- 3) 监控 CPU 100%，优化高延迟的操作。

## 6.总结回顾与作业实践



# 第 22 课总结回顾

Redis 基本功能

Redis 六大场景

Java、Spring 整合

Redis 高级功能

## 第 22 课作业实践

- 1、（选做）命令行下练习操作Redis的各种基本数据结构和命令。
- 2、（选做）分别基于 jedis, RedisTemplate, Lettuce, Redission 实现 redis 基本操作的 demo, 可以使用 spring-boot 集成上述工具。
- 3、（选做）spring 集成练习:
  - 1) 实现 update 方法, 配合 @CachePut
  - 2) 实现 delete 方法, 配合 @CacheEvict
  - 3) 将示例中的 spring 集成 Lettuce 改成 jedis 或 redisson。
- 4、（**必做**）基于 Redis 封装分布式数据操作:
  - 1) 在 Java 中实现一个简单的分布式锁;
  - 2) 在 Java 中实现一个分布式计数器, 模拟减库存。
- 5、基于 Redis 的 PubSub 实现订单异步处理

## 第 22 课作业实践

- 1、（挑战☆）基于其他各类场景，设计并在示例代码中实现简单 demo：
  - 1) 实现分数排名或者排行榜；
  - 2) 实现全局 ID 生成；
  - 3) 基于 Bitmap 实现 id 去重；
  - 4) 基于 HLL 实现点击量计数。
  - 5) 以 redis 作为数据库，模拟使用 lua 脚本实现前面课程的外汇交易事务。
- 2、（挑战☆☆）升级改造项目：
  - 1) 实现 guava cache 的 spring cache 适配；
  - 2) 替换 jackson 序列化为 fastjson 或者 fst, kryo；
  - 3) 对项目进行分析和性能调优。
- 3、（挑战☆☆☆）以 redis 作为基础实现上个模块的自定义 rpc 的注册中心。