

1.项目介绍

学习目标

- 了解系统的功能、背景、场景及项目要求
- 在架构角度思索系统可能面临的问题以及解决方案
- 学习本项目所涉及的中间件等基础知识
- 能够从0搭建springcloud微服务系统框架
- 能够完成编码，使用中间件完成系统的业务代码
- 学会部署上线，学会基于jenkins+docker swarm实现微服务的持续集成与动态扩容

1.1项目概述

1.1.1概述

假设公司要开年会，让你设计一套红包雨项目，在某段时间内随机发放不同的礼品，你该如何设计呢？

本项目实现了一个完整的红包雨模式抽奖系统，包括管理后台与前端界面。

由管理后台配置相关活动和奖品等信息，前端用户通过参与活动，完成抽奖。

1.1.2背景

1) 电商活动

互联网的发展中，电商是典型的应用场景。电商，即买卖，就如要抓住消费者的心理。那么各种促销、活动、成为电商公司必备的业务模块，尤其C端业务，面向普通大众用户。在所有活动中，抽奖是最典型的一种。

2) 红包雨

阿里春节的红包雨大家都多有参与。很多公司争相模仿，以实现随机派发红包的形式完成宣传与促销。红包雨可以理解为抽奖的一种特殊形式，奖品即红包。

3) 企业年会

年会基本是每个公司绕不开的话题。公司年会中的抽奖环节更是必不可少。尤其互联网公司，线上抽奖基本成为大家约定俗成的形式。打开手机，参与抽奖。正是本项目所涉及的现实应用场景。

1.1.3系统要求

1) 并发性

抽奖系统比如涉及到访问量大的问题。系统涉及所面临的第一关，即活动开始的瞬间，大批用户点击的涌入。怎样设计系统以达到如此高并发情况下的及时响应是本项目的重中之重。

2) 库存控制

抽奖面临的必然是奖品。数量控制是必须要做到精准吻合。不允许出现设置了5个奖品，最终6人中奖这种类似的问题出现。其中的本质是奖品库存的控制。后续的章节中会详细介绍本项目中如何做到控制库存的。

3) 投放策略

在活动时间段内，管理员设置好的一堆奖品如何投放？红包何时出现？年会奖品什么时候可以被抽中？这些都涉及到投放策略。

本项目中会给大家展示最常见的一种策略，即在活动时间内，奖品随机出现。最后的课程会给出引申，如何灵活扩展实现其他的投放算法。

4) 边界控制

活动何时开始？何时结束？倒计时如何控制。这涉及到活动的边界。开始前要提防用户提前进入抽奖。结束后要即使反馈结果给用户，告知活动已结束。

5) 活动自由配置

活动的配置由后台管理员完成，可以自由配置活动的开始结束时间，主题、活动简介、有哪些奖品、不同等级的用户中奖的策略。这就要求系统必须具备足够的业务灵活性。

6) 中奖策略

每个用户参与抽奖后，要遵从后台管理员所设定的中奖策略，典型的场景是针对用户设置最大中奖数。一旦用户中奖后，要进入计数，达到最大中奖数后，即使活动未结束，用户继续参与，也不能再让其中奖。而是将奖品机会倾向于其他参与者。下面的章节中会为大家展示如何根据后台策略精确控制用户中奖数量。

1.2 功能展示

1.2.1 管理后台

1) 会员管理

功能：用户查询、用户新增、删除、修改密码

用户管理为管理员提供基本的用户录入。本项目以企业年会为背景，可参与抽奖的用户由管理员后台直接录入，不允许私自注册其他非法账号。已录入的账号可以在抽奖前端页面中登录，参与抽奖。

在电商面向C端用户的情况下，新增一个注册接口，允许用户自行注册参与抽奖。

会员管理 x							
+ 增加 编辑 删除 修改密码 导出 搜索 重置							
	用户名	姓名	身份证号	手机号码	等级	注册时间	更新时间
1	<input type="checkbox"/> test	张艳新	15042619890518175	15201251947	普通会员	2019-09-27 02:38:42	2019-09-27 02:38:42
2	<input type="checkbox"/> 0	0	0	0	普通会员	2019-09-16 10:19:53	2019-09-16 10:19:53
3	<input type="checkbox"/> 4	4	4	4	四级会员	2019-09-16 10:19:40	2019-09-16 10:19:40
4	<input type="checkbox"/> 3	3	3	3	三级会员	2019-09-16 10:19:26	2019-09-16 10:19:26
5	<input type="checkbox"/> 2	2	2	2	二级会员	2019-09-16 10:19:12	2019-09-16 10:19:12
6	<input type="checkbox"/> 1	1	1	1	一级会员	2019-09-16 10:19:01	2019-09-16 10:19:01

2) 会员等级

功能：等级新增、删除、编辑

不同等级的会员有不同的中奖策略设置。比如高级别的会员中奖次数更多。详细会涉及下面活动配置中策略配置一节

首页 会员管理 x 等级管理 x				
等级名称: <input type="text"/>				
+ 增加 ✎ 编辑 ✕ 删除 📍 详情				
	<input type="checkbox"/>	等级代号	等级名称	创建时间
1	<input type="checkbox"/>	4	四级会员	2019-08-24 03:29:25
2	<input type="checkbox"/>	3	三级会员	2019-08-23 05:07:23
3	<input type="checkbox"/>	2	二级会员	2019-08-23 05:07:22
4	<input type="checkbox"/>	1	一级会员	2019-08-23 05:07:20
5	<input type="checkbox"/>	0	普通会员	2019-08-23 05:07:19

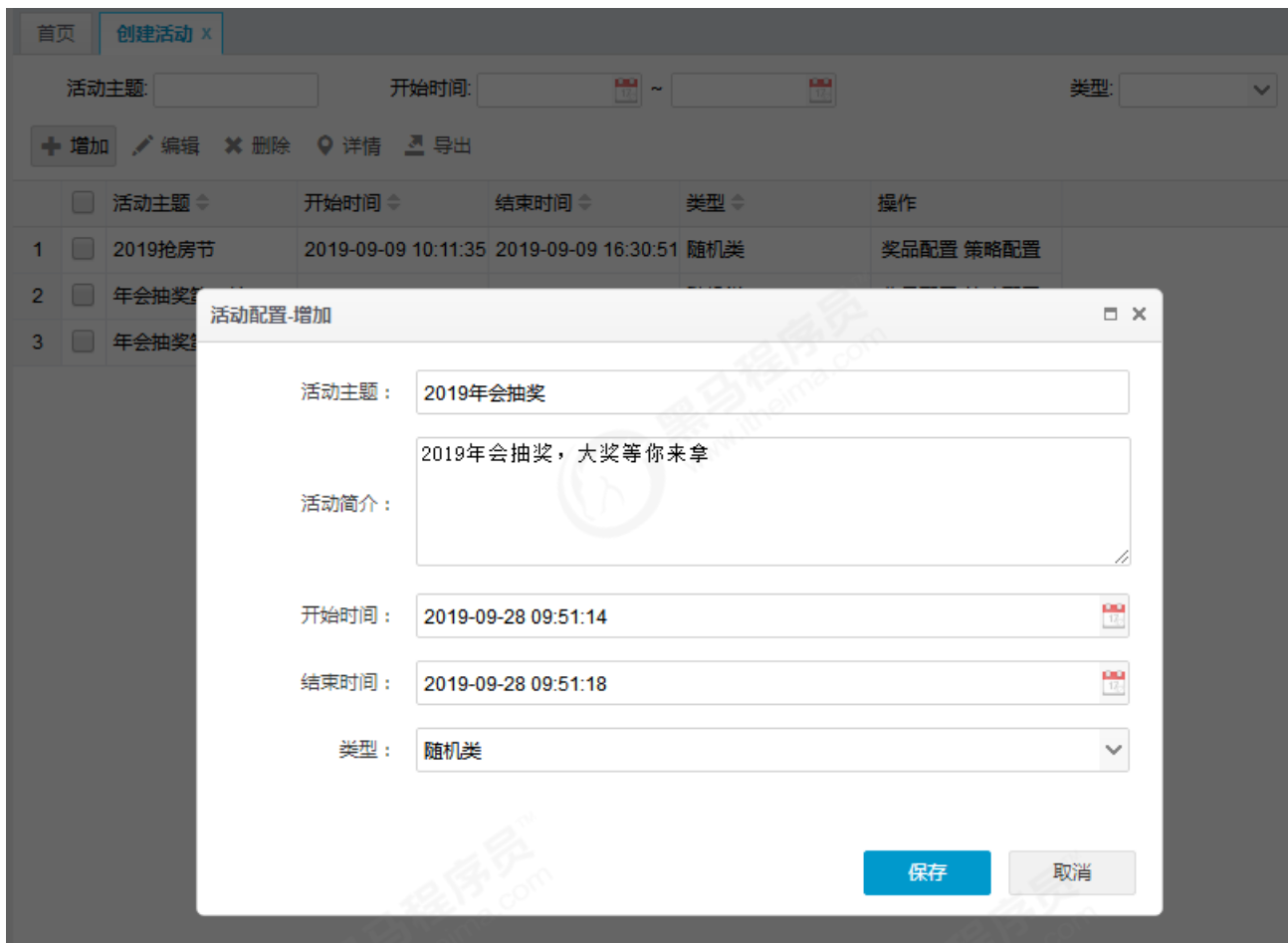
3) 活动管理

基础信息配置

功能：新增活动，修改活动，删除活动，配置活动基本信息（开始结束时间，标题，说明）

活动的基本信息管理功能

首页 会员管理 x 等级管理 x 创建活动 x						
活动主题: <input type="text"/> 开始时间: <input type="text"/> ~ <input type="text"/> 类型: <input type="text"/>						
+ 增加 ✎ 编辑 ✕ 删除 📍 详情 📄 导出 🔍 搜索 🔄 重置						
	<input type="checkbox"/>	活动主题	开始时间	结束时间	类型	操作
1	<input type="checkbox"/>	2019抢房节	2019-09-09 10:11:35	2019-09-09 16:30:51	随机类	奖品配置 策略配置
2	<input type="checkbox"/>	年会抽奖第二轮	2019-09-05 10:12:15	2019-08-29 00:00:00	随机类	奖品配置 策略配置
3	<input type="checkbox"/>	年会抽奖第一轮	2019-09-05 10:12:21	2019-08-29 00:00:00	随机类	奖品配置 策略配置



策略配置

功能：新增，修改，删除策略

策略涉及到用户的中奖次数，可以为不同等级的用户设置不同的最大中奖机会。不设置或者设置为0表示次数不限。



活动主题	开始时间	结束时间	类型	操作
2019抢房节	2019-09-09 10:11:35	2019-09-09 16:30:51	随机类	奖品配置 策略配置
年会抽奖第二轮	2019-09-05 10:12:15	2019-08-29 00:00:00	随机类	奖品配置 策略配置
年会抽奖第一轮	2019-09-05 10:12:21	2019-08-29 00:00:00	随机类	奖品配置 策略配置

活动	会员等级	可抽奖次数 (0为不限)	最大中奖次数 (0为不限)
2019抢房节	四级会员	0	100
2019抢房节	三级会员	1	2
2019抢房节	二级会员	31	0
2019抢房节	一级会员	2	0

奖品配置

功能：添加，删除，编辑奖品

为活动配置响应的奖品，可以添加多个不同的奖品，并为每个奖品设置单独的数量。

创建活动 x

活动主题: 开始时间: 结束时间: 类型: 操作

加 编辑 删除 详情 导出

Q 搜索 C 重置

活动主题	开始时间	结束时间	类型	操作
2019抢房节	2019-09-09 10:11:35	2019-09-09 16:30:51	随机类	奖品配置 策略配置
年会抽奖第二轮	2019-09-05 10:12:15	2019-08-29 00:00:00	随机类	奖品配置 策略配置
年会抽奖第一轮	2019-09-05 10:12:21	2019-08-29 00:00:00	随机类	奖品配置 策略配置

子页面

+ 增加 编辑 删除

活动	奖品	数量
1 2019抢房节	哈士奇一只	4
2 2019抢房节	二环内四合院一套	6
3 2019抢房节	地下车位一个	5
4 2019抢房节	美国商务办公室一套	5

4) 奖品管理

奖品管理

功能: 奖品增加, 编辑, 删除

录入奖品的基本信息, 可供多个活动引用。

首页 会员管理 x 等级管理 x 创建活动 x 奖品管理 x

奖品名称: 市场价: 搜索 重置

+ 增加 编辑 删除 详情 导出

	奖品名称	图片	简介	市场价
1	不长胖奶茶一杯		奶茶原为中国北方游牧	50
2	狗狗一只		简介: 中华田园犬, 性	1000
3	地下车位一个		地库设计导则 - 说明 ?	500
4	美国商务办公室一套		白宫 (英语: The Wh	1000
5	劳斯莱斯一辆		劳斯莱斯 (Rolls-Roy	5000000
6	二环内四合院一套		四合院, 又称四合房, 50000000	
7	哈士奇一只		西伯利亚雪橇犬是原	1000

5) 信息管理

中奖统计

功能: 只有按条件查询, 不涉及其他操作

统计每个活动的奖品总数, 以及被抽走的数量。该功能只涉及数据的统计, 不涉及新增修改删除, 属于只读操作。

首页 会员管理 x 等级管理 x 创建活动 x 奖品管理 x 中奖统计 x
活动主题: <input type="text"/> 开始时间: <input type="text"/> ~ <input type="text"/>
详情 导出 搜索 重置
<input type="checkbox"/> 活动主题 开始时间 结束时间 活动类型 总奖品数 已抽中
1 <input type="checkbox"/> 财务自由就靠这了 2019-09-16 16:41:46 2019-09-16 16:43:46 随机类 40
2 <input type="checkbox"/> 2019抢房节 2019-09-09 10:11:35 2019-09-09 16:30:51 随机类 20
3 <input type="checkbox"/> 年会抽奖第二轮 2019-09-05 10:12:15 2019-08-29 00:00:00 随机类 36
4 <input type="checkbox"/> 年会抽奖第一轮 2019-09-05 10:12:21 2019-08-29 00:00:00 随机类 14
5 <input type="checkbox"/> 10周年庆典 2019-09-24 15:48:12 2019-09-24 15:55:12 随机类 42 5

中奖列表

功能：基于各种条件查询中奖详情

可以根据所需条件，查询到相关的中奖信息，中奖人信息，奖品信息，中奖时间等。该功能只涉及数据的统计，不涉及新增修改删除，属于只读操作。

首页 会员管理 x 等级管理 x 创建活动 x 奖品管理 x 中奖统计 x 中奖列表 x
活动主题: <input type="text"/> 用户名: <input type="text"/> 手机号码: <input type="text"/> 奖品名称: <input type="text"/> 中奖时间: <input type="text"/> ~ <input type="text"/>
详情 导出 搜索 重置
<input type="checkbox"/> 活动主题 活动类型 用户名 姓名 身份证号 手机号码 会员等级 奖品名称 市场价 中奖时间
1 <input type="checkbox"/> 10周年庆典 随机类 2 2 2 2 二级会员 美国商务办公一套 1000 2019-09-24 15:27:11
2 <input type="checkbox"/> 10周年庆典 随机类 2 2 2 2 二级会员 二环内四合院一套 50000000 2019-09-24 15:27:11
3 <input type="checkbox"/> 10周年庆典 随机类 2 2 2 2 二级会员 哈士奇一只 1000 2019-09-24 15:27:10
4 <input type="checkbox"/> 10周年庆典 随机类 2 2 2 2 二级会员 iPhoneX一部 5000 2019-09-24 15:26:46
5 <input type="checkbox"/> 10周年庆典 随机类 2 2 2 2 二级会员 iPhoneX一部 5000 2019-09-24 15:26:45

6) 系统管理

操作日志

功能：查询管理员的操作日志

该功能用于记录管理员的操作。可以根据ip，操作时间内容，以及操作人查询到在后台中的行为。只涉及数据的统计，不涉及新增修改删除，属于只读操作。

首页 会员管理 x 等级管理 x 创建活动 x 奖品管理 x 中奖统计 x 中奖列表 x 操作日志 x					
用户: <input type="text"/> 操作内容: <input type="text"/> ip: <input type="text"/> 操作时间: <input type="text"/> ~ <input type="text"/>					
导出					
	<input type="checkbox"/>	用户	操作内容	ip	操作时间
1	<input type="checkbox"/>	test	登陆系统	113.44.46.127	2019-10-07 10:49:50
2	<input type="checkbox"/>	zcurd	登陆系统	14.131.252.229	2019-09-29 09:06:18
3	<input type="checkbox"/>	zcurd	登陆系统	14.131.250.25	2019-09-28 12:36:32
4	<input type="checkbox"/>	admin	登陆系统	14.131.250.25	2019-09-28 10:44:14
5	<input type="checkbox"/>	test	退出系统	14.131.250.25	2019-09-28 10:44:01
6	<input type="checkbox"/>	test	登陆系统	14.131.250.25	2019-09-28 09:27:36
7	<input type="checkbox"/>	zcurd	登陆系统	14.131.250.25	2019-09-27 21:15:44
8	<input type="checkbox"/>	test	登陆系统	14.131.250.25	2019-09-27 15:53:00
9	<input type="checkbox"/>	zcurd	登陆系统	14.131.250.25	2019-09-27 15:25:57
10	<input type="checkbox"/>	zcurd	[会员test] 增加	14.131.250.25	2019-09-27 10:38:42
11	<input type="checkbox"/>	zcurd	登陆系统	14.131.250.25	2019-09-27 10:36:49
12	<input type="checkbox"/>	zcurd	登陆系统	14.131.250.25	2019-09-27 10:10:40
13	<input type="checkbox"/>	zcurd	登陆系统	14.131.250.25	2019-09-26 17:29:19
14	<input type="checkbox"/>	test	登陆系统	14.131.250.25	2019-09-26 17:22:28
15	<input type="checkbox"/>	test	登陆系统	14.131.250.25	2019-09-25 11:25:37
16	<input type="checkbox"/>	zcurd	登陆系统	14.131.250.25	2019-09-25 10:47:27
17	<input type="checkbox"/>	test	登陆系统	14.131.250.25	2019-09-25 09:54:06

1.2.2前台展示

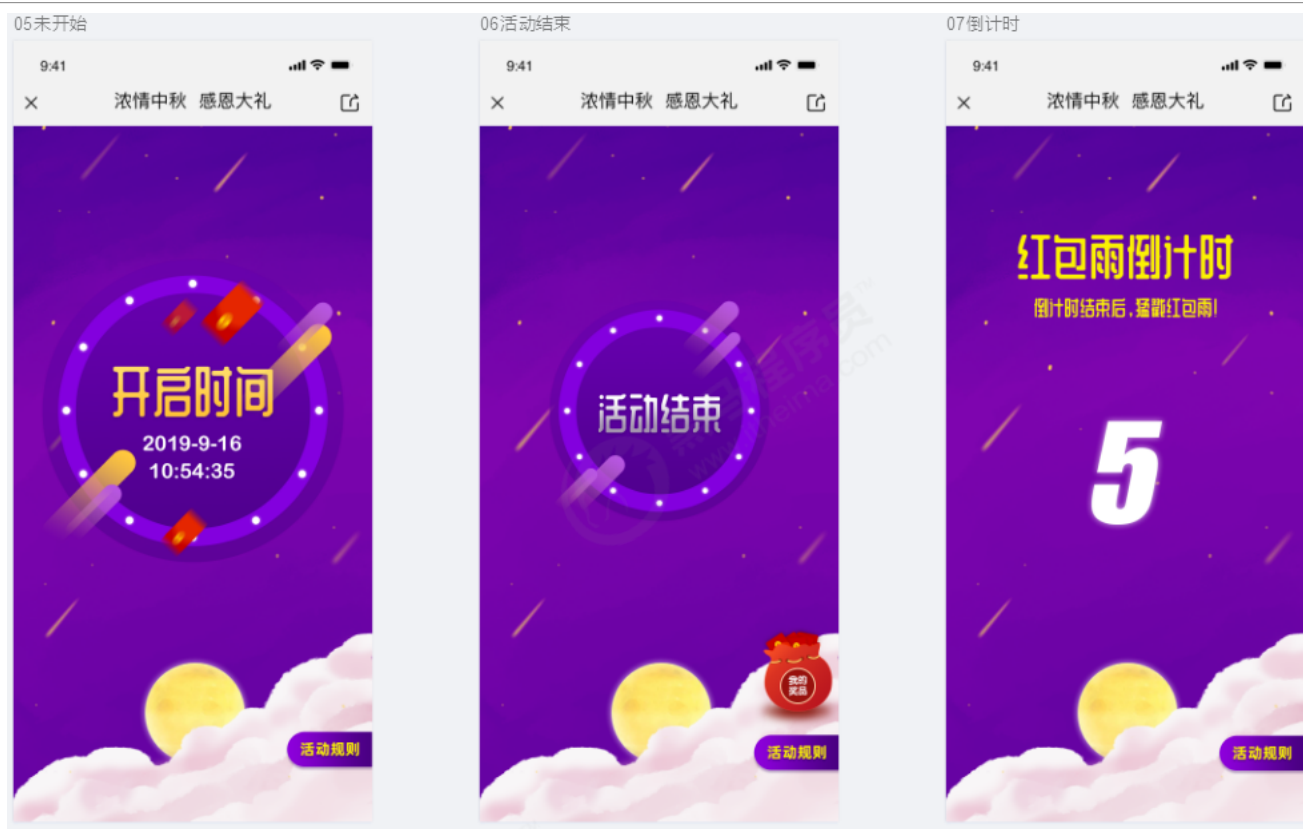
1) 活动列表



2) 活动详情



3) 抽奖展示



4) 个人中心

11我的



11我的_未中奖



14我的奖品_有中奖



15我的奖品_没中奖



1.3 中间件介绍

1.3.1 redis

1) 简介

Redis是当前比较热门的NOSQL系统之一，它是一个开源的使用ANSI c语言编写的**key-value**存储系统。

2) 应用场景

1. 缓存，这是Redis当今最为人熟知的使用场景。在提升服务器性能方面非常有效；
2. 排行榜，使用传统的关系型数据库（mysql oracle 等）来做这个事，非常的麻烦，而利用Redis的zset(有序集合)数据结构能够简单的搞定；
3. 计数器/限速器，利用Redis中原子性的自增操作，我们可以统计类似用户点赞数、用户访问数等，这类操作如果用MySQL，频繁的读写会带来相当大的压力；限速器比较典型的使用场景是限制某个用户访问某个API的频率，常用的有抢购时，防止用户疯狂点击带来不必要的压力，在本项目中会用到该功能；
4. 好友关系，利用集合的一些命令，比如求交集、并集、差集等。可以方便解决一些共同好友、共同爱好之类的功能；
5. 简单消息队列，除了Redis自身的发布/订阅模式，我们也可以利用List来实现一个队列机制，比如：到货通知、邮件发送之类的需求，不需要高可靠，但是会带来非常大的DB压力，完全可以用List来完成异步解耦；
6. Session共享，借助spring-session，后端用Redis保存Session后，无论用户落在那台机器上都能够获取到对应的Session信息。
7. 热数据查询，一些频繁被访问的数据，经常被访问的数据如果放在关系型数据库，每次查询的开销都会很大，而放在redis中，因为redis是放在内存中的，会得到量级的提升。

3) 数据类型

hset: key-field-value结构，用于一组相似类别的k-v值存储。类似java中的hashset工具

list: 队列，可以实现左右进出操作。类比java中的LinkedList，本项目中的抽奖令牌桶，使用的就是list

kv: 最典型的缓存存储结构，value可以存储对应的对象。

zset: 有序集合，在排序类，例如搜索词排名场景中会用到。

4) 相关命令

对KEY操作的命令

exists(key): 确认一个key是否存在 del(key): 删除一个key type(key): 返回值的类型 keys(pattern): 返回满足给定pattern的所有key randomkey: 随机返回key空间的一个 keyrename(oldname, newname): 重命名key dbsize: 返回当前数据库中key的数目 expire: 设定一个key的活动时间(s) ttl: 获得一个key的活动时间 move(key, dbindex): 移动当前数据库中的key到dbindex数据库 flushdb: 删除当前选择数据库中的所有key flushall: 删除所有数据库中的所有key

对String操作的命令

set(key, value): 给数据库中名称为key的string赋予值value get(key): 返回数据库中名称为key的string的value getset(key, value): 给名称为key的string赋予上一次的value mget(key1, key2,..., key N): 返回库中多个string的value setnx(key, value): 添加string，名称为key，值为value setex(key, time, value): 向库中添加string，设定过期时间time mset(key N, value N): 批量设置多个string的值 msetnx(key N, value N): 如果所有名称为key i的

string都不存在 incr(key): 名称为key的string增1操作 incrby(key, integer): 名称为key的string增加integer
decr(key): 名称为key的string减1操作 decrby(key, integer): 名称为key的string减少integer append(key,
value): 名称为key的string的值附加value substr(key, start, end): 返回名称为key的string的value的子串

对List操作的命令

rpush(key, value): 在名称为key的list尾添加一个值为value的元素 lpush(key, value): 在名称为key的list头添加一个值为value的元素 llen(key): 返回名称为key的list的长度 lrange(key, start, end): 返回名称为key的list中start至end之间的元素 ltrim(key, start, end): 截取名称为key的list index(key, index): 返回名称为key的list中index位置的元素 lset(key, index, value): 给名称为key的list中index位置的元素赋值 lrem(key, count, value): 删除count个key的list中值为value的元素 lpop(key): 返回并删除名称为key的list中的首元素 rpop(key): 返回并删除名称为key的list中的尾元素 blpop(key1, key2,... key N, timeout): lpop命令的block版本。brpop(key1, key2,... key N, timeout): rpop的block版本。

对Set操作的命令

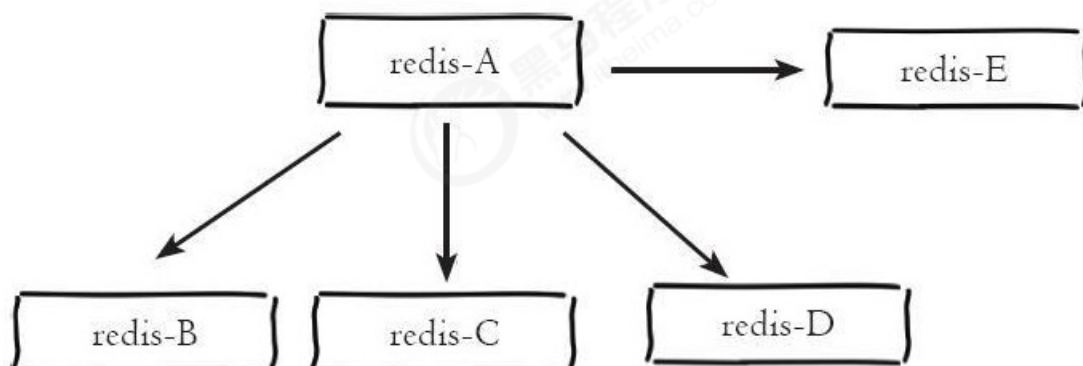
sadd(key, member): 向名称为key的set中添加元素member srem(key, member): 删除名称为key的set中的元素member spop(key): 随机返回并删除名称为key的set中一个元素 smove(srckey, dstkey, member): 移到集合元素 scard(key): 返回名称为key的set的基数 sismember(key, member): member是否是名称为key的set的元素 sinter(key1, key2,...key N): 求交集 sinterstore(dstkey, (keys)): 求交集并将交集保存到dstkey的集合 sunion(key1, (keys)): 求并集 sunionstore(dstkey, (keys)): 求并集并将并集保存到dstkey的集合 sdiff(key1, (keys)): 求差集 sdiffstore(dstkey, (keys)): 求差集并将差集保存到dstkey的集合 smembers(key): 返回名称为key的set的所有元素 srandmember(key): 随机返回名称为key的set的一个元素

对Hash操作的命令

hset(key, field, value): 向名称为key的hash中添加元素field hget(key, field): 返回名称为key的hash中field对应的value hmget(key, (fields)): 返回名称为key的hash中field i对应的value hmset(key, (fields)): 向名称为key的hash中添加元素field hincrby(key, field, integer): 将名称为key的hash中field的value增加integer hexists(key, field): 名称为key的hash中是否存在键为field的域 hdel(key, field): 删除名称为key的hash中键为field的域 hlen(key): 返回名称为key的hash中元素个数 hkeys(key): 返回名称为key的hash中所有键 hvals(key): 返回名称为key的hash中所有键对应的value hgetall(key): 返回名称为key的hash中所有的键(field)及其对应的value

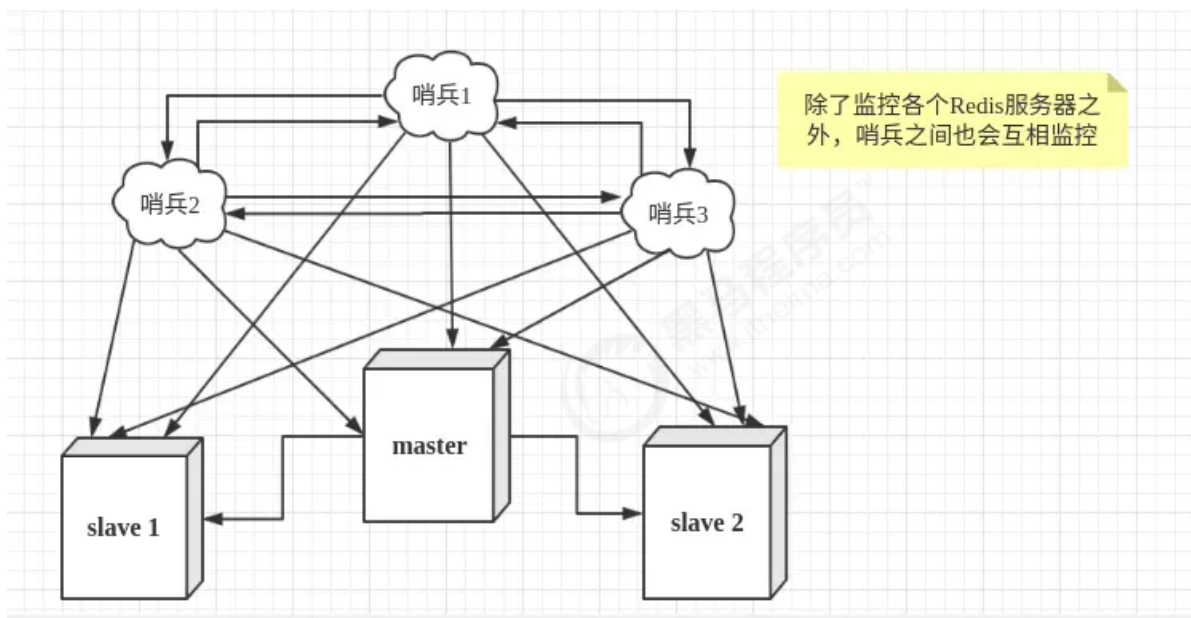
5) 三种模式

主从复制: 主从模式指的是使用一个redis实例作为主机，其余的实例作为备份机。主机和从机的数据完全一致，主机支持数据的写入和读取等各项操作，而从机则只支持与主机数据的同步和读取。主从模式很好的解决了数据备份问题，并且由于主从服务数据几乎是一致的，因而可以将写入数据的命令发送给主机执行，而读取数据的命令发送给不同的从机执行，从而达到读写分离的目的。



哨兵: 哨兵模式是一种特殊的模式，哨兵是一个独立的进程，独立运行。

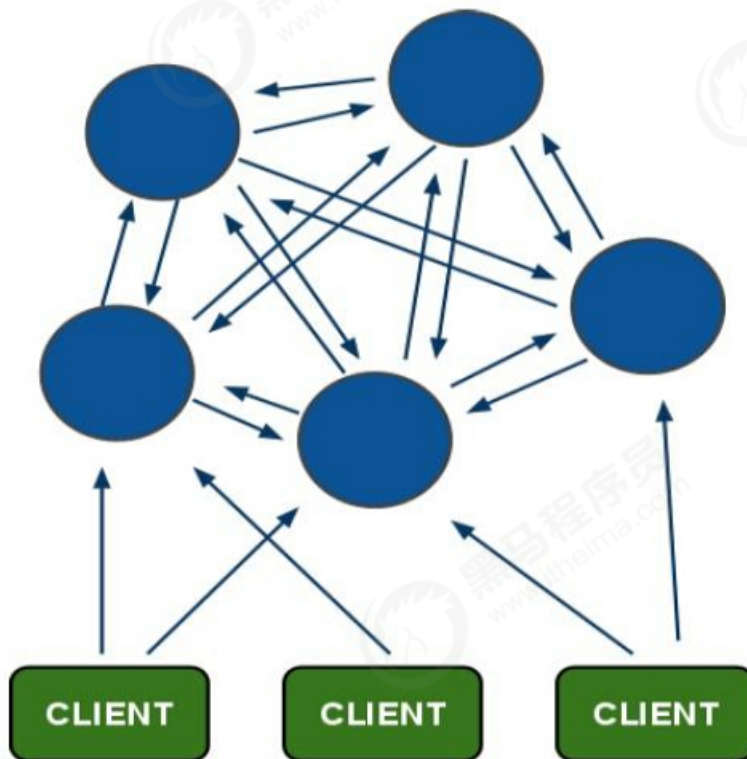
当哨兵检测到master宕机，会自动将slave切换成master，然后通过发布订阅模式通知其他的从服务器，修改配置文件，让它们切换主机。做到了主从自动切换和高可用。



集群: Redis Cluster是一个高性能高可用的分布式系统。由多个Redis实例组成的整体，数据按照Slot存储分布在多个Redis实例上，通过Gossip协议来进行节点之间通信。

Redis 3.0之后版本支持。

客户端连任意节点。



6)持久化

Redis支持RDB和AOF两种持久化机制，持久化功能有效地避免因进程退出造成的数据丢失问题，当下次重启时利用之前持久化文件即可实现数据恢复。

rdb:

在指定时间间隔内，将内存中的数据快照写入磁盘，也就是Snapshot快照，它恢复时是将快照文件直接读到内存中，来达到恢复数据的，rdb也是redis默认的持久化方式。

Redis会单独创建(fork)一个子进程来进行持久化，会先将数据写进一个临时文件中，等到持久化过程结束了，再用这个临时文件替换上次持久化好的文件。在这个过程中，只有子进程来负责IO操作，主进程仍然处理客户端的请求，这就确保了极高的性能。

优点

适合大规模数据恢复的场景，数据紧凑，易迁移

缺点 RDB这种持久化方式数据完整性很难保证，虽然我们可以用过修改持久化的频率，但是如果还没有触发快照时，本机就宕机了，那么对数据库所做的写操作就丢失了。每次进行RDB时，父进程都会fork一个子进程，由子进程来进行实际的持久化操作，数据量大时，那么fork出子进程的这个过程将是非常耗时的。

aof:

以日志的形式记录Redis每一个写操作，将Redis执行过的所有写指令记录下来，注意，读操作是不需要记录的，redis启动之后会读取appendonly.aof文件，将之前的操作复现，来完成恢复数据的工作。

appendfsync always:每修改同步，每一次发生数据变更都会持久化到磁盘上，性能较差，但数据完整性较好。

appendfsync everysec: 每秒同步，每秒内记录操作，异步操作，如果一秒内宕机，有数据丢失。

appendfsync no:不同步。

重写: 当然如果AOF文件一直被追加，这就可能导致AOF文件过于庞大。因此，为了避免这种状况，Redis新增了重写机制，当AOF文件的大小超过所指定的阈值时，Redis会自动启用AOF文件的内容压缩，只保留可以恢复数据的最小指令集

优点

看上去轻量化，增量形式

保留了redis的历史操作

多种策略配置，相对灵活

缺点

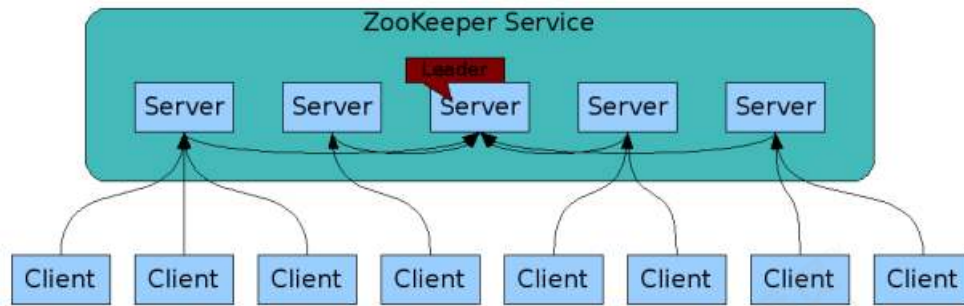
对于相同的数据集来说，AOF文件要比RDB文件大。

根据所使用的持久化策略来说，AOF的速度要慢与RDB。

1.3.2 zookeeper

1) 简介

zookeeper是一个分布式服务框架，是Apache Hadoop的一个子项目，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。



2) 节点类型

临时节点: 临时节点的生命周期和客户端会话绑定在一起，客户端会话失效，则这个节点就会被自动清除。

永久节点: 该数据节点被创建后，就会一直存在于zookeeper服务器上，直到有删除操作来主动删除这个节点。

3) 使用场景

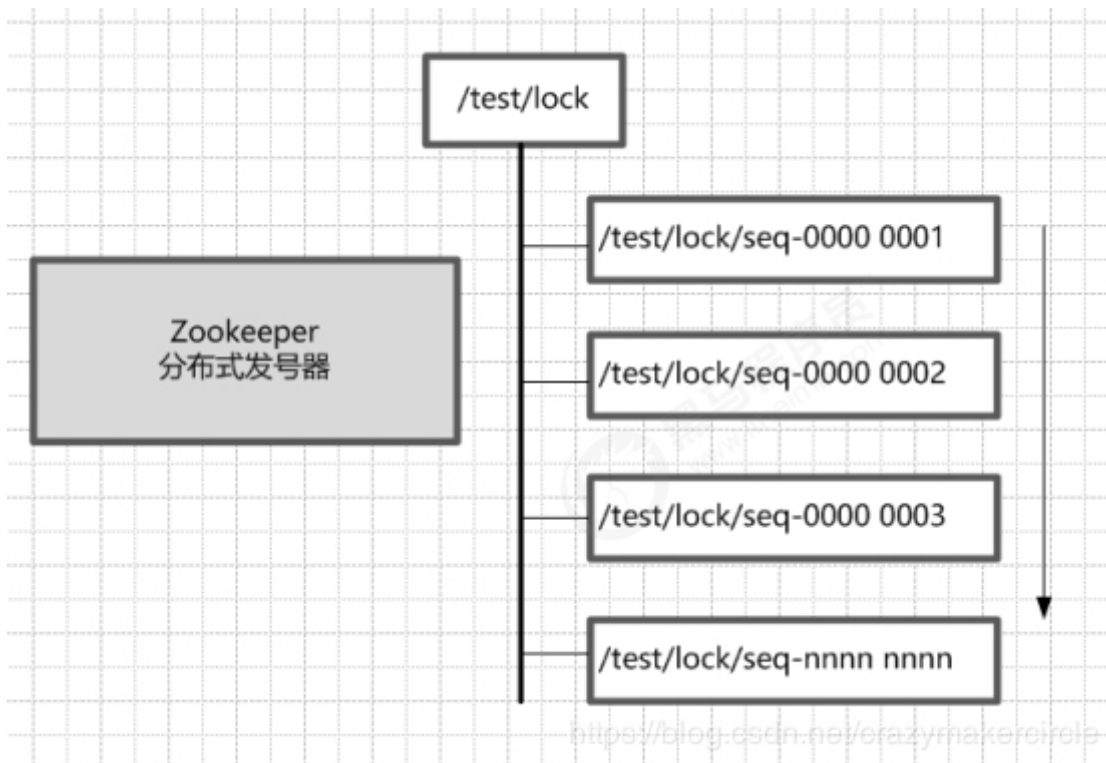
配置中心: 配置中心，顾名思义就是将配置数据写到ZK节点上，供各个分布式机器获取配置，同时监听自己对应的节点。实现配置信息的集中式管理和动态更新。

命名服务: 在分布式系统中，通过使用命名服务，客户端应用能够根据指定名字来获取资源或服务的地址，提供者等信息。被命名的实体通常可以是集群中的机器，提供的服务地址，远程对象等等，这些我们都可以统称他们为名字（Name）。通过调用ZK提供的创建节点的API，能够很容易创建一个全局唯一的path，这个path就可以作为一个名称。

分布式通知: ZooKeeper的watcher注册与异步通知机制，能够很好的实现分布式环境下不同系统之间的通知与协调，实现对数据变更的实时处理。不同系统都对ZK上同一个znode进行注册，监听znode的变化（包括znode本身内容及子节点的），其中一个系统update了znode，那么另一个系统能够收到通知，并作出相应处理

选主: 利用ZooKeeper的一致性，能够保证在分布式高并发情况下节点创建的全局唯一性，即：同时有多个客户端请求创建 /currentMaster 节点，最终一定只有一个客户端请求能够创建成功。利用这个特性，就能很轻易的在分布式环境中进行集群选取了。

分布式锁: 分布式锁，这个主要得益于ZooKeeper的节点创建和事件监听机制。



4) 相关命令

创建节点 create 列出节点 ls 获取节点信息 get 检查状态 stat 修改节点 set 删除节点 rmr 删除节点 delete

5) 高可用

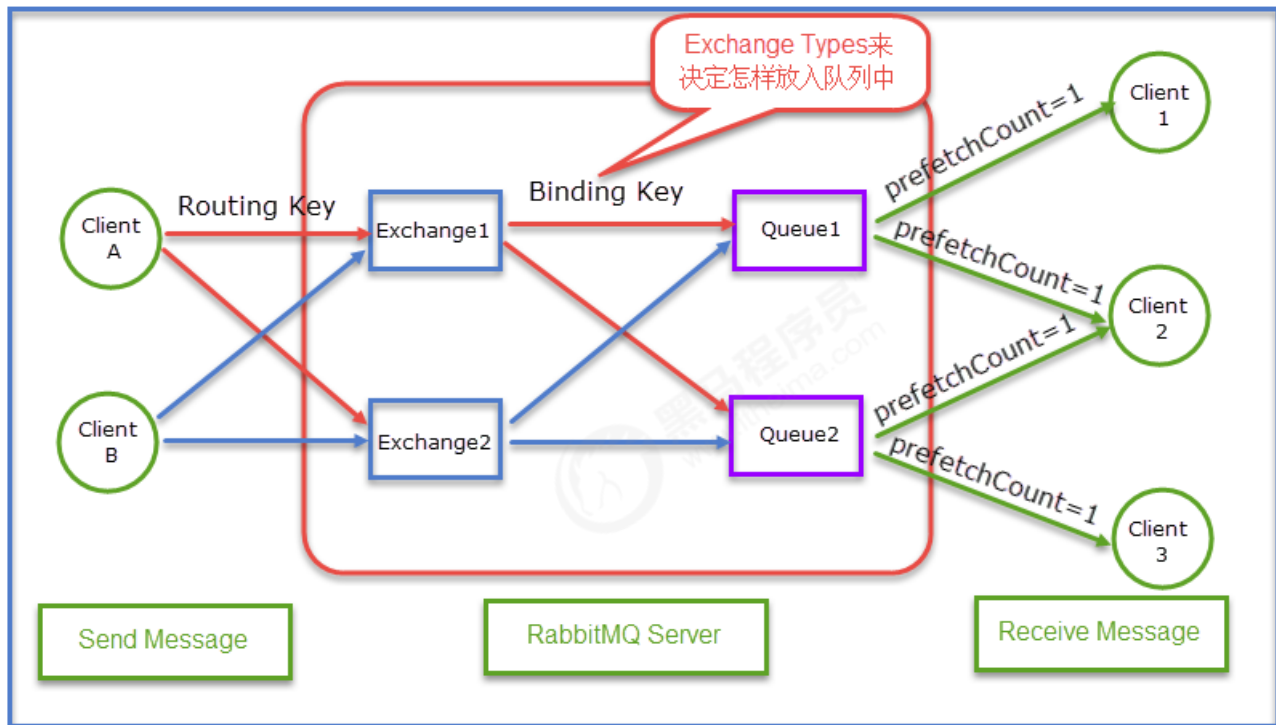
集群与选主: 以5台机器启动时场景为主, 过程如下:

1) 服务器1启动, 此时只有它一台服务器启动了, 它发出去的报没有任何响应, 所以它的选举状态一直是LOOKING状态。2) 服务器2启动, 它与最开始启动的服务器1进行通信, 互相交换自己的选举结果, 由于两者都没有历史数据, 所以id值较大的服务器2胜出, 但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是3), 所以服务器1, 2还是继续保持LOOKING状态。3) 服务器3启动, 根据前面的理论分析, 服务器3成为服务器1, 2, 3中的老大, 而与上面不同的是, 此时有三台服务器选举了它, 所以它成为了这次选举的leader。4) 服务器4启动, 根据前面的分析, 理论上服务器4应该是服务器1, 2, 3, 4中最大的, 但是由于前面已经有半数以上的服务器选举了服务器3, 状态是following, 所以它只能接收当小弟的命了。5) 服务器5启动, 同4一样, 当小弟。

1.3.3 rabbitmq

1) 简介

RabbitMQ是一个由erlang开发的AMQP (Advanced Message Queue) 的开源实现



2) 模块介绍

Broker: 可以简单理解为一台物理机器。

Producer: 消息生产者，就是投递消息的程序。

Consumer: 消息消费者，就是接受消息的程序。

Exchange: 消息交换机，它指定消息按什么规则，路由到哪个队列。fanout, direct, topic, header

Queue: 消息的载体，每个消息都会被投到一个或多个队列。

Binding: 绑定，它的作用就是把exchange和queue按照路由规则绑定起来。

Routing Key: 路由关键字，exchange根据这个关键字进行消息投递。

vhost: 虚拟主机，一个broker里可以有多个vhost，用作不同用户的权限分离。（不涉及）

3) 应用

消息传递、异步处理、应用解耦、流量削峰

4) 高可用

发送方: confirm机制（发送成功后有异步通知）

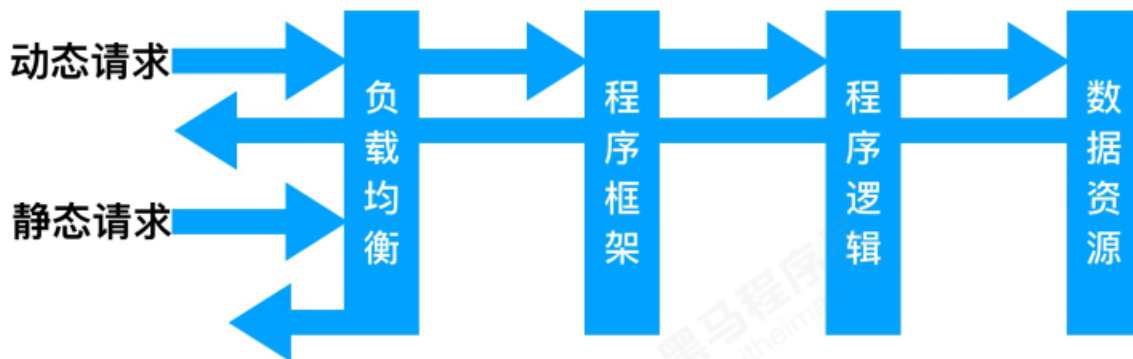
消费端: ACK消息应答机制

rabbit: queue持久化，消息持久化（deliveryMode=2）

1.3.4 nginx

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力在同类型的网页服务器中几乎成为公认的标杆，在百度、京东、新浪、网易、腾讯、淘宝等互联网公司中均有应用。

1) 动静分离



静态资源：由nginx作为web服务器身份，直接返回

动态资源：nginx将请求转发出去，交给后端应用服务器处理

2) 负载均衡

当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群提升并行处理能力。并且多台服务器可以平均分担负载，不会因为某台服务器负载高宕机而某台服务器闲置的情况。这时使用nginx实现了机器之间的负载均衡。

3) 配置介绍

location

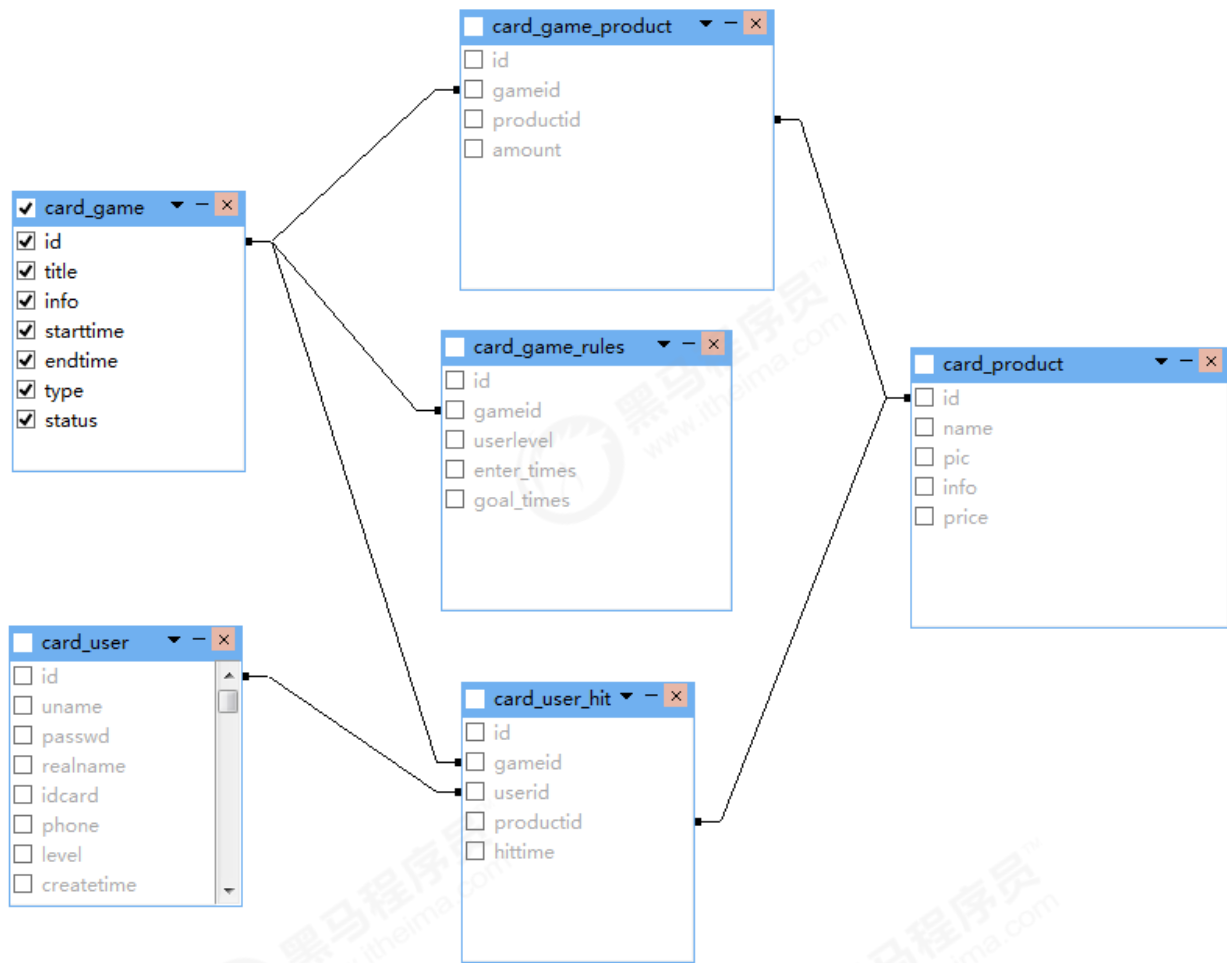
proxy_pass

upstream

2.系统设计

2.1建模

2.1.1 ER图



2.1.2 数据表

1) 奖品表

card_product

字段	类型	备注
id	int(10) unsigned	
name	varchar(255)	奖品名称
pic	varchar(255)	图片
info	varchar(1000)	简介
price	decimal(10,2)	市场价

2) 活动表

card_game

字段	类型	备注
id	int(10) unsigned	
title	varchar(255)	活动主题
info	varchar(1000)	活动简介
starttime	datetime	开始时间
endtime	datetime	结束时间
type	tinyint(2)	类型（1=概率类，2=随机类）
status	tinyint(1)	状态（0=新建，1=已加载）

3) 会员表

card_user

字段	类型	备注
id	int(11) unsigned	
uname	varchar(20)	用户名
passwd	varchar(50)	密码
realname	varchar(10)	姓名
idcard	varchar(18)	身份证号
phone	varchar(15)	手机号码
level	smallint(6)	等级
createtime	datetime	注册时间
updatetime	datetime	更新时间

4) 策略表

card_game_rules

字段	类型	备注
id	int(11) unsigned	
gameid	int(11) unsigned	活动id
userlevel	smallint(6)	会员等级
enter_times	smallint(6)	可抽奖次数（0为不限）
goal_times	smallint(6)	最大中奖次数（0为不限）

5) 中奖纪录

card_user_hit

字段	类型	备注
id	int(10) unsigned	
gameid	int(10) unsigned	活动
userid	int(10) unsigned	用户
productid	int(10) unsigned	奖品
hittime	datetime	中奖时间

6) 奖品活动关联关系

card_game_product

字段	类型	备注
id	int(10) unsigned	
gameid	int(11) unsigned	活动id
productid	int(11)	奖品id
amount	smallint(6)	数量

2.1.3 视图

1) 中奖信息

view_card_user_hit

字段	类型	备注
id	int(10) unsigned	
title	varchar(255)	活动主题
type	varchar(100)	值
uname	varchar(20)	用户名
realname	varchar(10)	姓名
idcard	varchar(18)	身份证号
phone	varchar(15)	手机号码
level	varchar(100)	值
name	varchar(255)	奖品名称
price	decimal(10,2)	市场价
gameid	int(10) unsigned	活动
userid	int(10) unsigned	用户
productid	int(10) unsigned	奖品
hittime	datetime	中奖时间

2) 奖品数统计

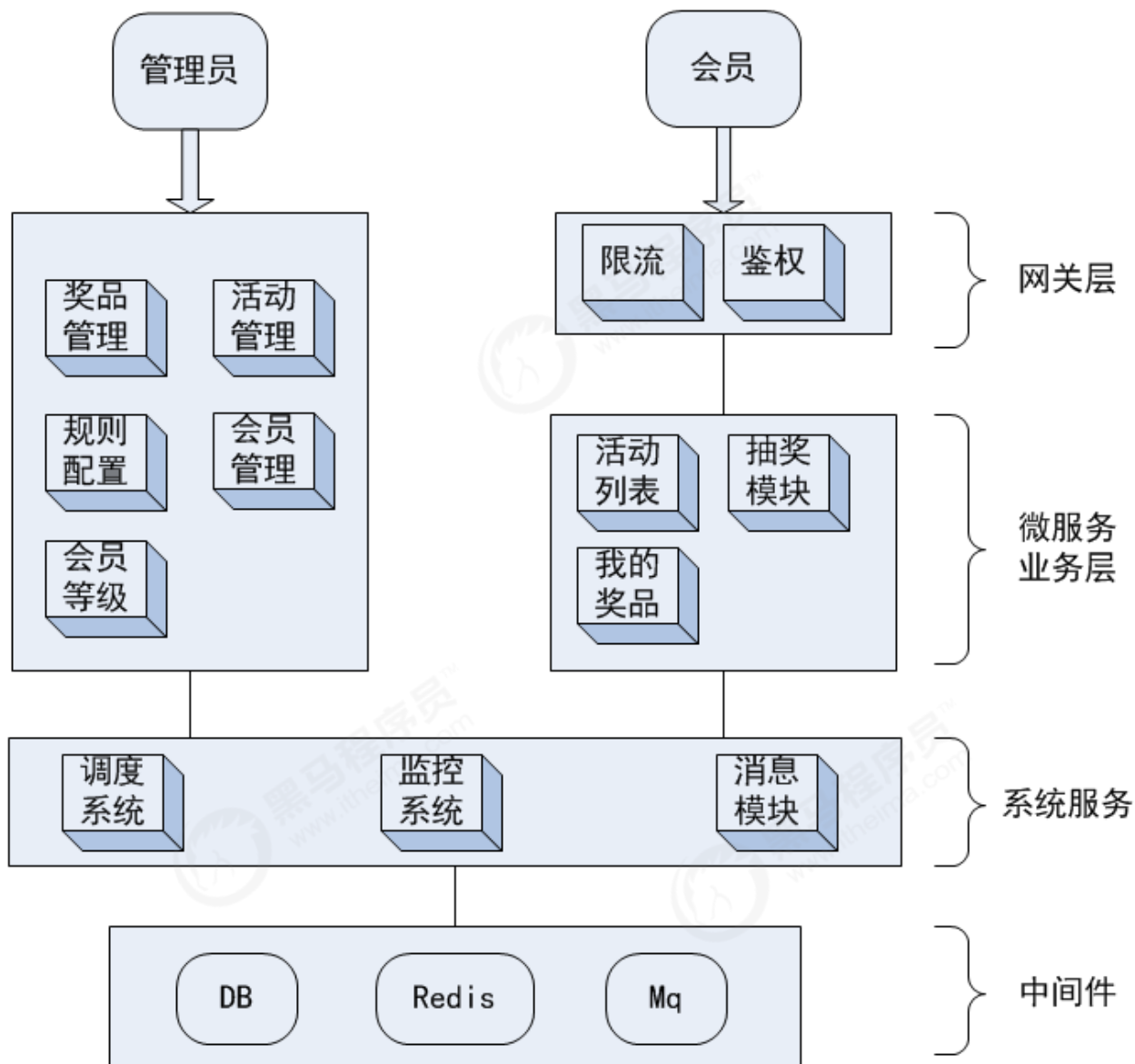
view_game_curinfo

字段	类型	备注
id	int(10) unsigned	
title	varchar(255)	活动主题
starttime	datetime	开始时间
endtime	datetime	结束时间
type	varchar(100)	值
total	decimal(27,0)	
hit	bigint(21)	

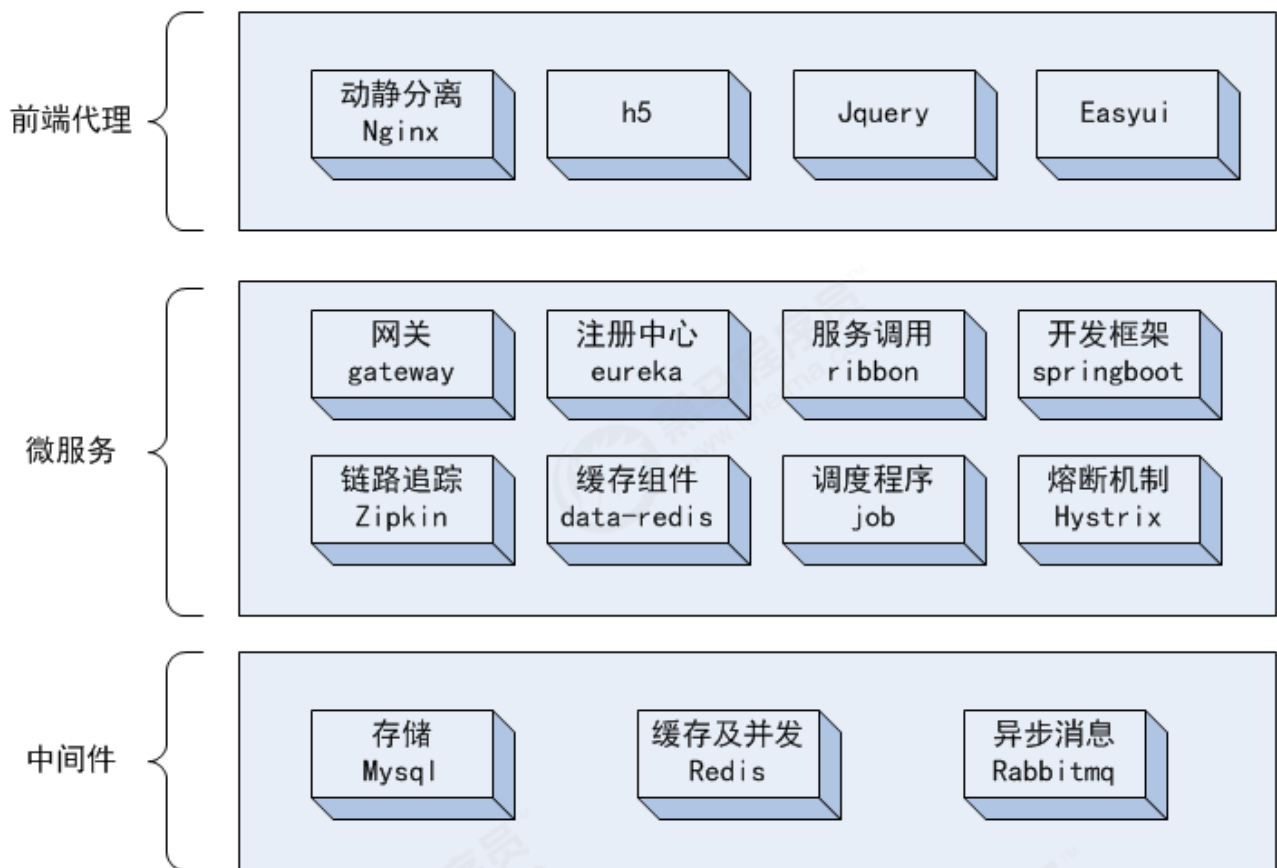
2.2 概要设计

2.2.1 系统拓扑

1) 业务架构



2) 软件架构



2.2.2 设计原则

1) 动静分离

1·静态文件分离，nginx直接响应，不要再绕后台应用机器

2) 微服务化

1·将模块细粒度拆分，微服务化

2·借助docker swarm的容器管理功能，实现不同服务的副本部署，滚动更新

3·在本项目中，api模块就部署了3份，以适应前端的高并发

3) 负载均衡

1·多个实例之间通过nginx做负载均衡，提升并发性能

2·本项目为大家展示的模块均部署在1台节点。生产环境涉及多台机器，用upstream实现。

4) 异步消息

1·中奖后，中奖人及奖品信息要持久化到数据库。引入rabbitmq，将抽奖操作与数据库操作异步隔离。

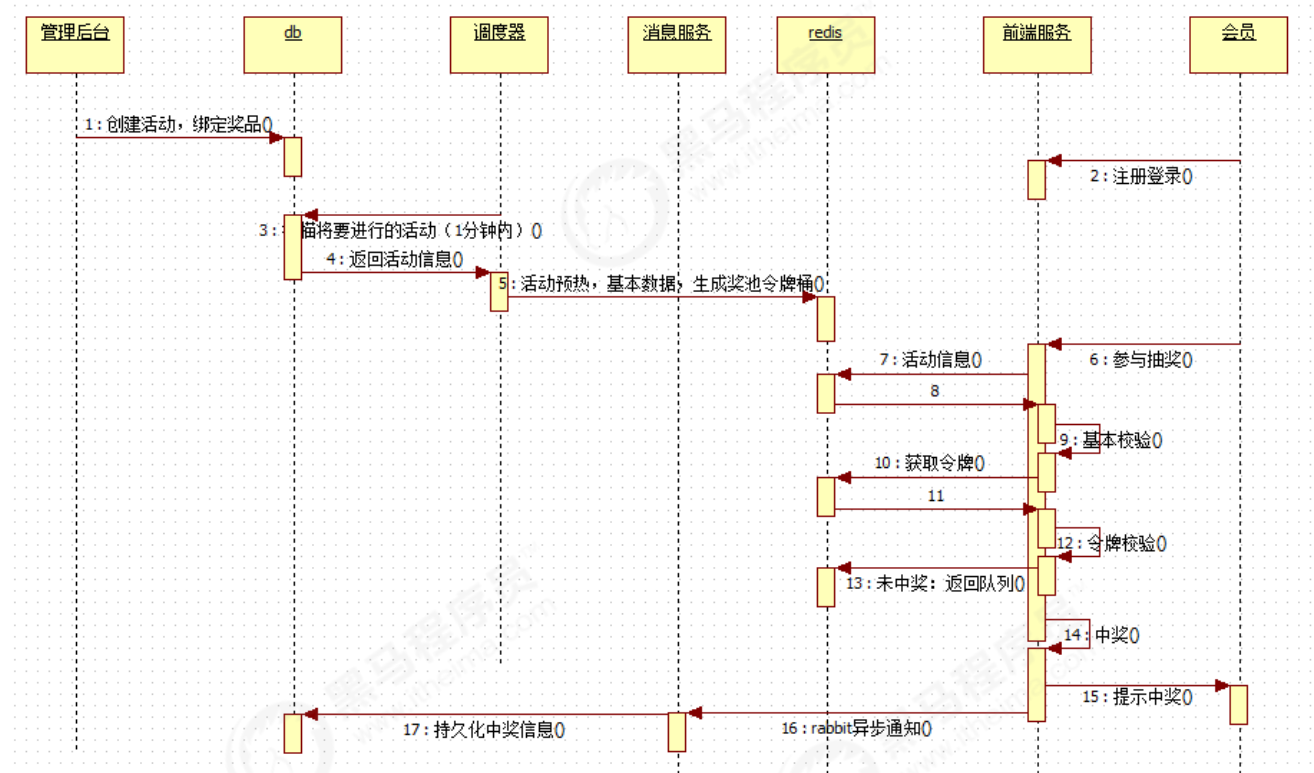
2·抽奖中奖后，只需要将中奖信息放入rabbitmq，并立即返回中奖信息给前端用户。

3·后端msg模块消费rabbitmq消息，缓慢处理。

5) 缓存预热

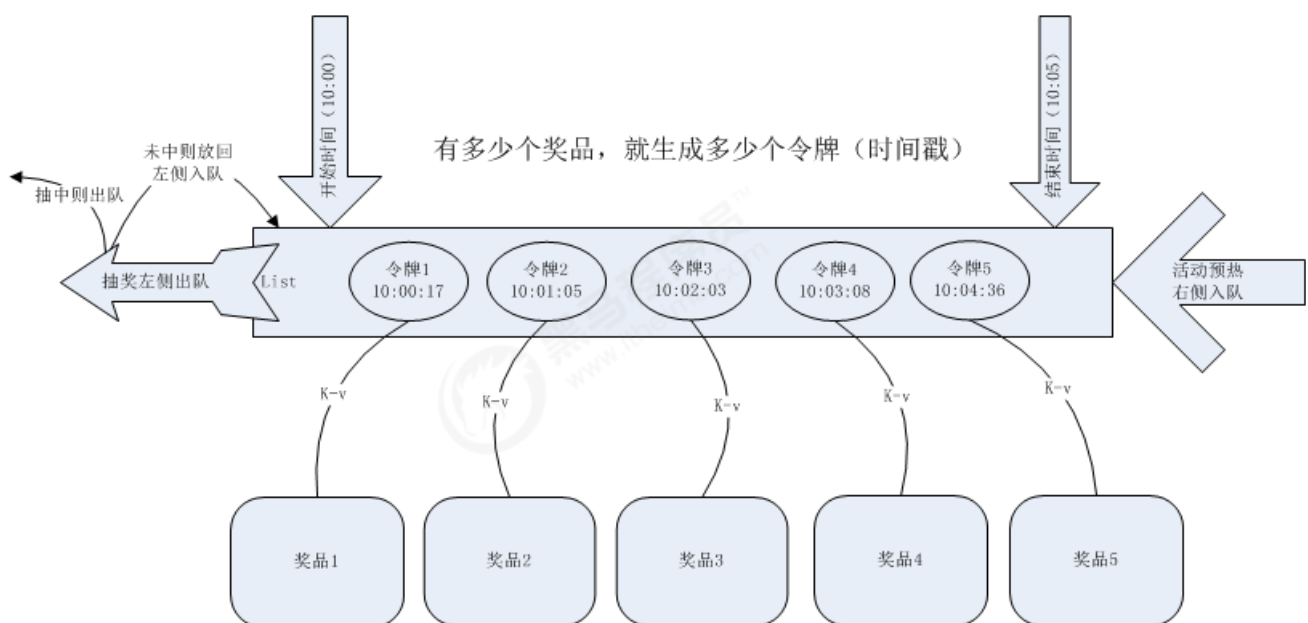
1. 每隔1分钟扫描一次活动表，查询未来1分钟内将要开始的活动。
2. 将扫到的活动加载进redis，包括活动详细信息，中奖策略信息，奖品信息，抽奖令牌。
3. 活动正式开始后，基于redis数据做查询，不必再与数据库打交道。

2.2.3交互序列图



2.2.4缓存体系

缓存体系概览图:



1) 活动基本信息

k-v, 以活动id为key, 活动对象为value, 永不超时

```
redisUtil.set(RedisKeys.INFO+game.getId(),game,-1);
```

2) 活动策略信息

hset, 以活动id为group, 用户等级为key, 策略值为value

```
redisUtil.hset(RedisKeys.MAXGOAL + game.getId(),r.getUserlevel()+"",r.getGoalTimes());  
redisUtil.hset(RedisKeys.MAXENTER + game.getId(),r.getUserlevel()+"",r.getEnterTimes());
```

3) 抽奖令牌桶

双端队列, 以活动id为key, 在活动时间段内, 随机生成时间戳做令牌, 有多少个奖品就生成多少个令牌。令牌即奖品发放的时间点。从小到大排序后从右侧入队。

```
redisUtil.rightPushAll(RedisKeys.TOKENS + game.getId(),tokenList);
```

4) 奖品映射信息

k-v, 以活动id_令牌为key, 奖品信息为value, 会员获取到令牌后, 如果令牌有效, 则用令牌token值, 来这里获取奖品详细信息

```
redisUtil.set(RedisKeys.TOKEN + game.getId()  
+"_"+token,productMap.get(cgp.getProductid()),expire);
```

5) 令牌设计技巧

假设活动时间间隔太短, 奖品数量太多。那么极有可能产生的时间戳发生重复。

解决技巧: 额外再附加一个随机因子。将 (时间戳 * 1000 + 3位随机数) 作为令牌。抽奖时, 将抽中的令牌/1000, 还原真实的时间戳。

```
//活动持续时间 (ms)  
long duration = end - start;  
long rnd = start + new Random().nextInt((int)duration);  
//为什么乘1000, 再额外加一个随机数呢? - 防止时间段奖品多时重复  
long token = rnd * 1000 + new Random().nextInt(999);
```

6) 中奖计数

k-v, 以活动id_用户id作为key, 中奖数为value, 利用redis原子性, 中奖后incr增加计数。

```
redisUtil.incr(RedisKeys.USERHIT+gameid+"_"+user.getId(),1);
```

7) 中奖逻辑判断

抽奖时, 从令牌桶左侧出队和当前时间比较, 如果令牌时间戳小于等于当前时间, 令牌有效, 表示中奖。大于当前时间, 则令牌无效, 将令牌还回, 从左侧压入队列。

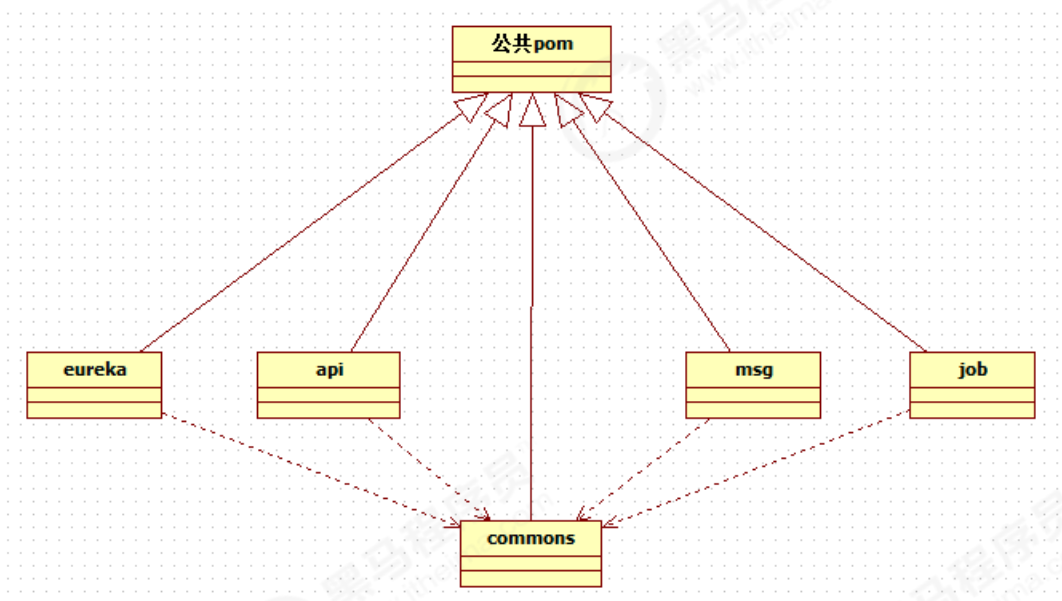
2.3 框架选型

2.3.1 管理后台

借助开源zcurd开发平台，完成后台基本的增删改查。

2.3.2 前台模块

基于springcloud构建微服务体系



1) 公共pom

所有项目共用的依赖及版本定义。继承自springboot 2.1.7.RELEASE，依赖cloud Greenwich.SR2

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.7.RELEASE</version>
</parent>
```

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-dependencies</artifactId>
  <version>Greenwich.SR2</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

2) 公共模块commons

分页、密码、统一结果dto等工具类，mybatis生成的实体，mapper，redis及rabbit的配置bean

```
▼ commons
  ▼ src
    ▼ main
      ▼ java
        ▼ cn.itcast.prize.commons
          ▼ config
            RabbitConfig
            RabbitKeys
            RedisConfig
            RedisKeys
          ▼ db
            > entity
            > mapper
          ▼ utils
            ApiResult
            PageBean
            PasswordUtil
            RedisUtil
```

3) 注册中心eureka

cloud官方推荐的微服务注册中心，所有的服务模块将注册在eureka中。

4) 调度模块job

集成当当网开源的elastic-job，活动的调度扫描预热等定时任务在该模块中。

5) 消息模块msg

rabbitmq消息消费端，用户中奖后将中奖信息丢入rabbitmq，msg模块将消费消息，将中奖信息写入db。

6) 接口模块api

提供给前端页面的rest接口，如：抽奖接口，活动查询接口，中奖信息接口等

2.3.3 微服务框架集成

1) 集成zookeeper配置中心

cloud配置中心使用zookeeper，在zookeeper中的目录节点以job项目为例：

- 引入zookeeper配置中心pom坐标

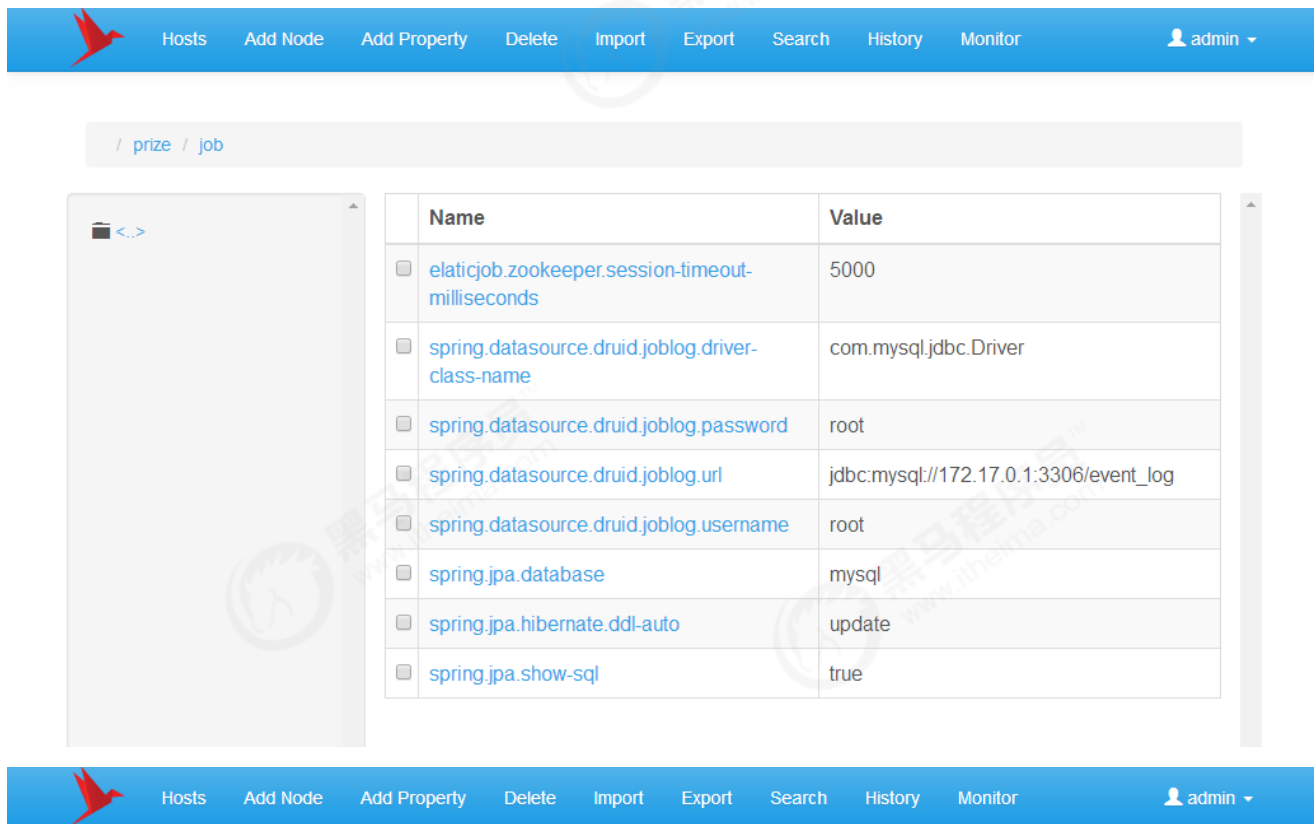
```
<!-- 配置中心-->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zookeeper-config</artifactId>
</dependency>
```

- 在job项目中放置bootstrap.properties文件，内容如下：

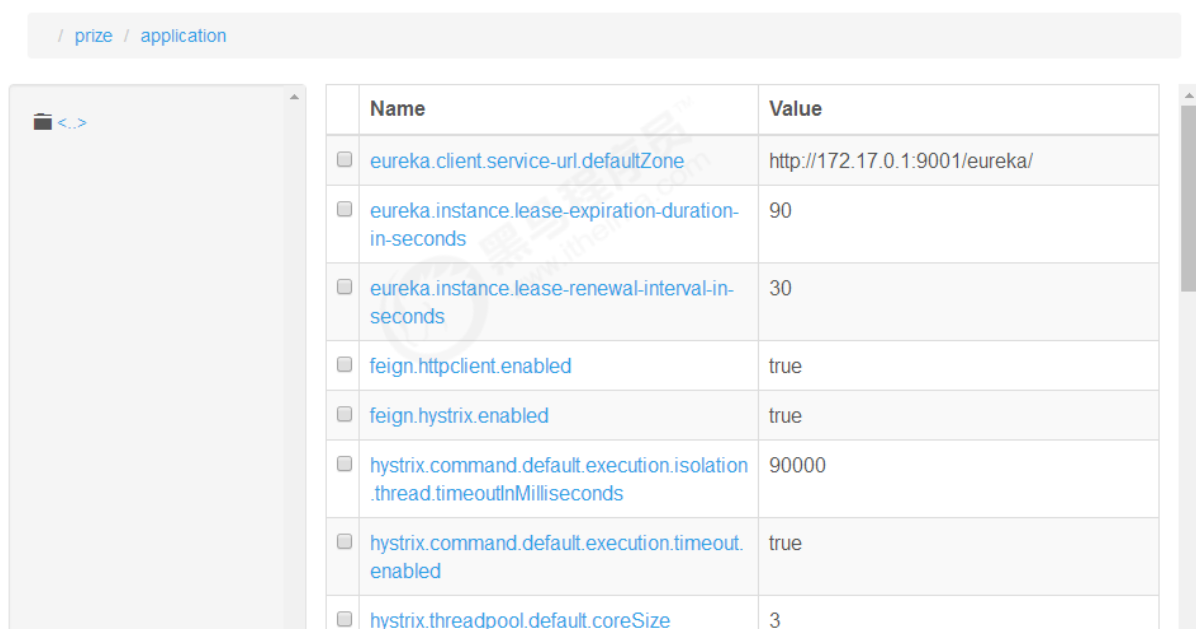

```
## 配置应用名称
spring.application.name=job
## 配置zookeeper地址
spring.cloud.zookeeper.connect-string=192.168.193.128:2181
## 在zookeeper中，配置存放路径的前缀
spring.cloud.zookeeper.config.root = prize
## 配置文件在zookeeper中的路径即： /prize/job
```

- 配置查找的优先级如下：

首先查找/prize/job目录，如果找不到，查 /prize/application目录，如果存在同名的配置优先级 job>application



Name	Value
elasticjob.zookeeper.session-timeout-milliseconds	5000
spring.datasource.druid.joblog.driver-class-name	com.mysql.jdbc.Driver
spring.datasource.druid.joblog.password	root
spring.datasource.druid.joblog.url	jdbc:mysql://172.17.0.1:3306/event_log
spring.datasource.druid.joblog.username	root
spring.jpa.database	mysql
spring.jpa.hibernate.ddl-auto	update
spring.jpa.show-sql	true



Name	Value
eureka.client.service-url.defaultZone	http://172.17.0.1:9001/eureka/
eureka.instance.lease-expiration-duration-in-seconds	90
eureka.instance.lease-renewal-interval-in-seconds	30
feign.httpclient.enabled	true
feign.hystrix.enabled	true
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds	90000
hystrix.command.default.execution.timeout.enabled	true
hystrix.threadpool.default.coreSize	3

2) 集成redis

坐标：

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

相关配置：

```
/prize/application=spring.redis.database=0
/prize/application=spring.redis.host=172.17.0.1
/prize/application=spring.redis.password=
/prize/application=spring.redis.pool.max-active=3
/prize/application=spring.redis.pool.max-idle=10
/prize/application=spring.redis.pool.max-wait=1000
/prize/application=spring.redis.pool.min-idle=1
/prize/application=spring.redis.port=6379
/prize/application=spring.redis.timeout=3000
```

3) 集成elastic-job

坐标：注意，curator版本会报冲突，排除后，手动引入版本



```
<properties>
    <job.version>2.1.5</job.version>
    <curator.version>2.10.0</curator.version>
</properties>

<!-- 框架核心jar包 -->
<dependency>
    <groupId>com.dangdang</groupId>
    <artifactId>elastic-job-lite-core</artifactId>
    <version>${job.version}</version>
    <exclusions>
        <exclusion>
            <artifactId>curator-client</artifactId>
            <groupId>org.apache.curator</groupId>
        </exclusion>
        <exclusion>
            <artifactId>curator-framework</artifactId>
            <groupId>org.apache.curator</groupId>
        </exclusion>
        <exclusion>
            <artifactId>curator-recipes</artifactId>
            <groupId>org.apache.curator</groupId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>${curator.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-client</artifactId>
    <version>${curator.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>${curator.version}</version>
</dependency>

<dependency>
    <groupId>com.dangdang</groupId>
    <artifactId>elastic-job-lite-spring</artifactId>
    <version>${job.version}</version>
</dependency>

<!-- 添加数据相关的驱动主要是为了记录任务相关的一些数据，日志 -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
</dependency>

<dependency>
```



```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-autoconfigure</artifactId>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

配置:

```
/prize/job=elasticjob.zookeeper.session-timeout-milliseconds=5000
/prize/job=spring.datasource.druid.joblog.driver-class-name=com.mysql.jdbc.Driver
/prize/job=spring.datasource.druid.joblog.password=root
/prize/job=spring.datasource.druid.joblog.url=jdbc:mysql://172.17.0.1:3306/event_log
/prize/job=spring.datasource.druid.joblog.username=root
/prize/job=spring.jpa.database=mysql
/prize/job=spring.jpa.hibernate.ddl-auto=update
/prize/job=spring.jpa.show-sql=true
```

4) 集成rabbitmq

坐标:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

配置:

```
/prize/application=spring.rabbitmq.host=172.17.0.1
/prize/application=spring.rabbitmq.password=guest
/prize/application=spring.rabbitmq.port=5672
/prize/application=spring.rabbitmq.publisher-confirms=true
/prize/application=spring.rabbitmq.username=guest
/prize/application=spring.rabbitmq.virtual-host=
```

5) 集成mysql数据源druid

坐标:



```
<!--数据源-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.31</version>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
    <version>1.1.10</version>
</dependency>
```

配置:

```
/prize/application=spring.datasource.connectionProperties=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000
/prize/application=spring.datasource.driver-class-name=com.mysql.jdbc.Driver
/prize/application=spring.datasource.filters=stat,wall,log4j
/prize/application=spring.datasource.initialSize=5
/prize/application=spring.datasource.maxActive=20
/prize/application=spring.datasource.maxPoolPreparedStatementPerConnectionSize=20
/prize/application=spring.datasource.maxWait=60000
/prize/application=spring.datasource.minEvictableIdleTimeMillis=300000
/prize/application=spring.datasource.minIdle=5
/prize/application=spring.datasource.password=root
/prize/application=spring.datasource.poolPreparedStatements=true
/prize/application=spring.datasource.testOnBorrow=false
/prize/application=spring.datasource.testOnReturn=false
/prize/application=spring.datasource.testWhileIdle=true
/prize/application=spring.datasource.timeBetweenEvictionRunsMillis=60000
/prize/application=spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
/prize/application=spring.datasource.url=jdbc:mysql://172.17.0.1/prize?
useUnicode=true&characterEncoding=utf-8
/prize/application=spring.datasource.useGlobalDataSourceStat=true
/prize/application=spring.datasource.username=root
/prize/application=spring.datasource.validationQuery=SELECT 'x'
```

6) 集成mybatis

坐标:

```
<!--mybatis-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>1.3.2</version>
</dependency>
```

配置:

```
/prize/application=mybatis.mapper-locations=classpath:mapper/*.xml
```

7) 集成session

坐标:

```
<dependency>
  <groupId>org.springframework.session</groupId>
  <artifactId>spring-session-data-redis</artifactId>
</dependency>
```

配置:

```
/prize/application=spring.session.store-type=redis
```

2.3.4 工具

1) zkui

zkui是zookeeper的一个管理工具，具有登录界面，及zk中节点和值的crud操作入口，支持导入导出功能。

zkui是一个jar包，可以用java -jar启动

配置:



```
#Server Port
serverPort=9090

#Comma seperated list of all the zookeeper servers
zkServer=192.168.193.128:2181

#Http path of the repository. Ignore if you dont intent to upload files from repository.
scmRepo=http://myserver.com/@rev1=

#Path appended to the repo url. Ignore if you dont intent to upload files from repository.
scmRepoPath=//appconfig.txt

#if set to true then userSet is used for authentication, else ldap authentication is used.
ldapAuth=false
ldapDomain=mycompany,mydomain

#ldap authentication url. Ignore if using file based authentication.
ldapUrl=ldap://<ldap_host>:<ldap_port>/dc=mycom,dc=com

#Specific roles for ldap authenticated users. Ignore if using file based authentication.
ldapRoleSet={"users": [{ "username":"domain\\user1" , "role": "ADMIN" }]}
userSet = {"users": [{ "username":"admin" , "password":"admin","role": "ADMIN" },{
"username":"appconfig" , "password":"appconfig","role": "USER" }]}

#Set to prod in production and dev in local. Setting to dev will clear history each time.
env=prod
jdbcClass=org.h2.Driver
jdbcUrl=jdbc:h2:zkui
jdbcUser=root
jdbcPwd=manager

#If you want to use mysql db to store history then comment the h2 db section.
#jdbcClass=com.mysql.jdbc.Driver
#jdbcUrl=jdbc:mysql://localhost:3306/zkui
#jdbcUser=root
#jdbcPwd=manager
loginMessage=Please login using admin/manager or appconfig/appconfig.
#session timeout 5 mins/300 secs.
sessionTimeout=60000

#Default 5 seconds to keep short lived zk sessions. If you have large data then the read will
take more than 30 seconds so increase this accordingly.
#A bigger zkSessionTimeout means the connection will be held longer and resource consumption
will be high.
zkSessionTimeout=50

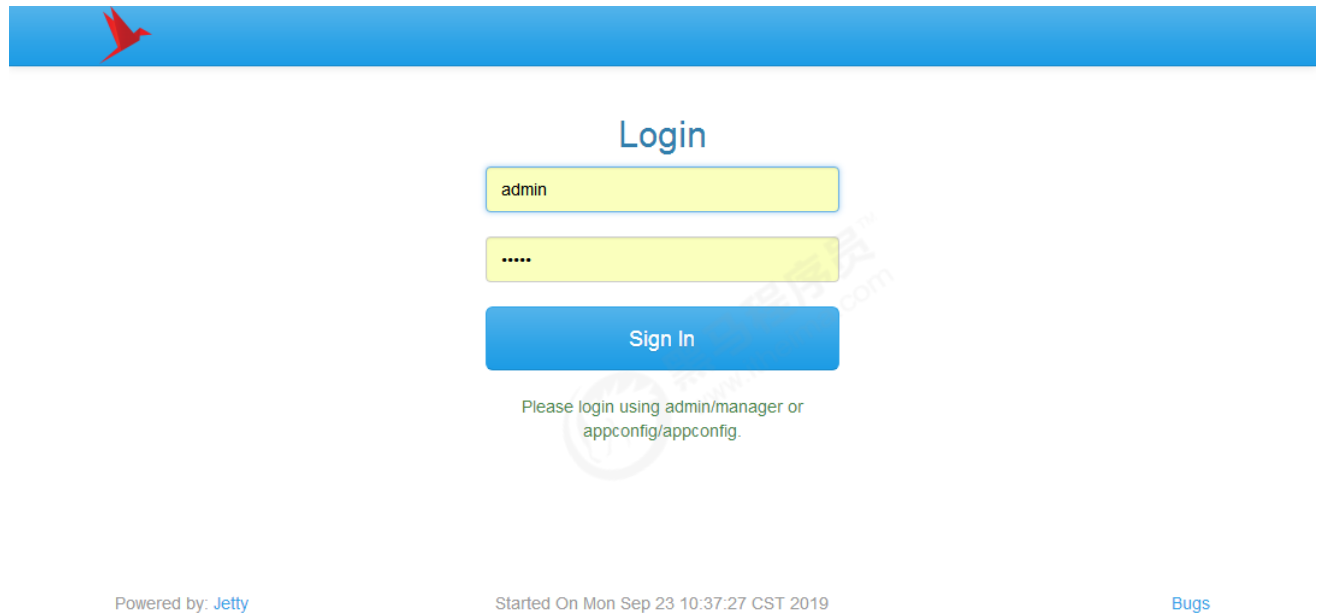
#Block PWD exposure over rest call.
blockPwdOverRest=false

#ignore rest of the props below if https=false.
https=false
keystoreFile=/home/user/keystore.jks
keystorePwd=password
keystoreManagerPwd=password

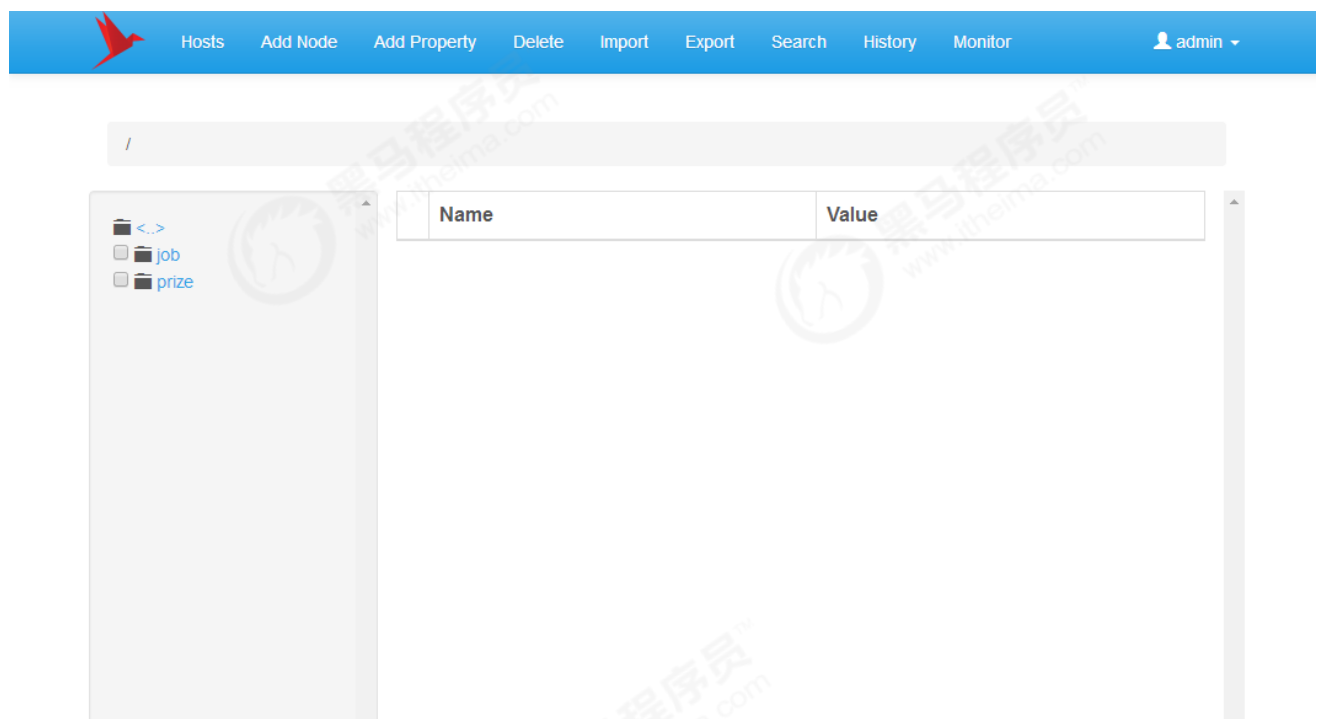
# The default ACL to use for all creation of nodes. If left blank, then all nodes will be
universally accessible
# Permissions are based on single character flags: c (Create), r (read), w (write), d (delete),
a (admin), * (all)
# For example defaultAcl={"acls": [{ "scheme":"ip", "id":"192.168.1.192", "perms":"*"},
{ "scheme":"ip", id":"192.168.1.0/24", "perms":"r"}]}
defaultAcl=

# Set X-Forwarded-For to true if zkui is behind a proxy
X-Forwarded-For=false
```


启动界面:



The login interface features a blue header bar with a red bird icon on the left. Below the header, the word "Login" is centered. There are two yellow input fields: the first contains the text "admin", and the second contains five dots. Below these fields is a blue "Sign In" button. A message below the button reads: "Please login using admin/manager or appconfig/appconfig." At the bottom of the page, there is a status bar with three items: "Powered by: Jetty", "Started On Mon Sep 23 10:37:27 CST 2019", and a link to "Bugs".



The dashboard interface has a blue header bar with a red bird icon on the left and a menu on the right. The menu items are: Hosts, Add Node, Add Property, Delete, Import, Export, Search, History, Monitor, and a user profile dropdown showing "admin". Below the header, there is a light gray sidebar on the left with a tree view containing "job" and "prize". The main content area has a table with two columns: "Name" and "Value". The table is currently empty.

2) mybatis-generator

引入坐标:

```
<dependency>
  <groupId>org.mybatis.generator</groupId>
  <artifactId>mybatis-generator-core</artifactId>
  <version>1.3.7</version>
</dependency>
```

配置pom插件:

```
<plugin>
  <groupId>org.mybatis.generator</groupId>
  <artifactId>mybatis-generator-maven-plugin</artifactId>
  <version>1.3.7</version>
  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.31</version>
    </dependency>
    <dependency>
      <groupId>com.itheima.prize</groupId>
      <artifactId>coder</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </dependency>
  </dependencies>
  <configuration>
    <overwrite>true</overwrite>
    <outputDirectory>${project.build.directory}</outputDirectory>
    <verbose>true</verbose>
    <tableNames>%</tableNames>
  </configuration>
</plugin>
```

配置表映射关系:



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
    PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
    "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">

<generatorConfiguration>
    <context id="MysqlContext" targetRuntime="MyBatis3" defaultModelType="flat">
        <property name="beginningDelimiter" value="`"/>
        <property name="endingDelimiter" value="`"/>

        <!-- 指定生成的java文件的编码,没有直接生成到项目时中文可能会乱码 -->
        <property name="javaFileEncoding" value="UTF-8"/>

        <!-- 生成的pojo, 将implements Serializable -->
        <plugin type="org.mybatis.generator.plugins.SerializablePlugin"></plugin>
        <!-- 这里的type里写的是你的实现类的类全路径 -->
        <!--<commentGenerator type="com.MyCommentGenerator">-->
            <!--<property name="suppressDate" value="true"/>-->
        <!--</commentGenerator>-->

        <commentGenerator>
            <property name="suppressDate" value="true"/>
        </commentGenerator>

        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
            connectionURL="jdbc:mysql://192.168.193.128/prize?tinyInt1isBit=false"
            userId="root"
            password="root">
        </jdbcConnection>

        <javaTypeResolver type="com.JavaTypeResolverDefaultImpl">
        </javaTypeResolver>

        <!-- entity 包路径 -->
        <javaModelGenerator targetPackage="com.itheima.prize.commons.db.entity"
targetProject="MAVEN">
            <property name="trimStrings" value="true"/>
        </javaModelGenerator>

        <!-- xml 路径-->
        <sqlMapGenerator targetPackage="com.itheima.prize.commons.db.xml.mapper"
targetProject="MAVEN"/>

        <javaClientGenerator type="MIXEDMAPPER"
targetPackage="com.itheima.prize.commons.db.mapper" targetProject="MAVEN"/>

        <table tableName="%">
            <generatedKey column="id" sqlStatement="Mysql" identity="true"/>
        </table>
    </context>
</generatorConfiguration>
```

tinyint会映射为bit，习惯上使用Integer更方便。自己定义一个type映射类并配置到上面的config.xml中

```
public class JavaTypeResolverDefaultImpl implements JavaTypeResolver {
    ....

    public JavaTypeResolverDefaultImpl() {
        ....
        this.typeMap.put(-6, new JavaTypeResolverDefaultImpl.JdbcTypeInfo("TINYINT", new
        FullyQualifiedJavaType(Integer.class.getName())));
        ....
    }

    ....
}
```

3) 分页PageHelper

坐标：

```
<!--mybatis分页插件-->
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
</dependency>
```

定义一个pagebean:

```
@ApiModel("分页信息")
public class PageBean<T> {
    @ApiModelProperty(value = "当前页, 1开始")
    private Integer currentPage = 1;
    @ApiModelProperty(value = "每页条数, 默认10")
    private Integer pageSize = 10;
    @ApiModelProperty(value = "总条数")
    private Long totalNum;
    @ApiModelProperty(value = "是否有下一页")
    private Integer isMore;
    @ApiModelProperty(value = "总页数")
    private Integer totalPage;
    @ApiModelProperty(value = "开始索引")
    private Integer startIndex;
    @ApiModelProperty(value = "本页数据")
    private List<T> items;

    ...
}
```

使用：

```
// 在查询之前，调用startPage，设置页码及条数
```

```
PageHelper.startPage(curpage, limit);
```

```
List<ViewCardUserHit> all = hitMapper.selectByExample(example);
```

查询的sql会被自动拼接limit

```
2019-10-07 16:39:58.988 DEBUG 1 --- [io-8080-exec-10] c.i.p.c.d.m.V.selectByExample_COUNT : ==> Preparing: SELECT count(0) FROM view_card_user_hit WHERE (gameid = ?)
2019-10-07 16:39:58.988 DEBUG 1 --- [io-8080-exec-10] c.i.p.c.d.m.V.selectByExample_COUNT : ==> Parameters: 1(Integer)
2019-10-07 16:39:58.989 DEBUG 1 --- [io-8080-exec-10] c.i.p.c.d.m.V.selectByExample_COUNT : <== Total: 1
2019-10-07 16:39:58.989 DEBUG 1 --- [io-8080-exec-10] c.i.p.c.d.m.V.selectByExample : ==> Preparing: select id, title, type, uname, realname, idcard, phone, lev
hittime from view_card_user_hit WHERE ( gameid = ? ) limit ?,?
2019-10-07 16:39:58.989 DEBUG 1 --- [io-8080-exec-10] c.i.p.c.d.m.V.selectByExample : ==> Parameters: 1(Integer), 0(Integer), 2(Integer)
2019-10-07 16:39:58.992 DEBUG 1 --- [io-8080-exec-10] c.i.p.c.d.m.V.selectByExample : <== Total: 2
```

返回的数据体:



```
{
  "code": 1,
  "msg": "成功",
  "data": {
    "currentPage": 1,
    "pageSize": 2,
    "totalNum": 5,
    "isMore": 1,
    "totalPage": 3,
    "startIndex": 0,
    "items": [
      {
        "id": 1,
        "title": "10周年庆典",
        "type": "随机类",
        "uname": "2",
        "realname": "2",
        "idcard": "2",
        "phone": "2",
        "level": "二级会员",
        "name": "iPhoneX一部",
        "price": 5000,
        "gameid": 1,
        "userid": 8,
        "productid": 1,
        "hittime": "2019-09-24 15:26:45"
      },
      {
        "id": 2,
        "title": "10周年庆典",
        "type": "随机类",
        "uname": "2",
        "realname": "2",
        "idcard": "2",
        "phone": "2",
        "level": "二级会员",
        "name": "iPhoneX一部",
        "price": 5000,
        "gameid": 1,
        "userid": 8,
        "productid": 1,
        "hittime": "2019-09-24 15:26:46"
      }
    ]
  }
}
```

4) swagger2

坐标:



```
<!--swagger2增强，官方ui太low， 访问地址： /doc.html -->
<dependency>
  <groupId>com.github.xiaoymin</groupId>
  <artifactId>swagger-bootstrap-ui</artifactId>
  <version>1.8.8</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
```

配置：



```
package com.itheima.prize.api.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SwaggerConfig {

    @Value("${spring.application.name}")
    private String appName;

    @Bean
    public Docket docket() {
        return new Docket(DocumentationType.SWAGGER_2).apiInfo(apiInfo()).select()
//        当前包路径
        .apis(RequestHandlerSelectors.basePackage("com.itheima.prize.api.action"))
        .paths(PathSelectors.any()).build();

    }

    //构建api文档的详细信息函数
    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            //页面标题
            .title("抽奖系统前端"+appName)
            //联系人
            .contact(new Contact("Shawn", null, "xxx@itcast.cn"))
            //版本号
            .version("1.0")
            //描述
            .description("提供给前端页面调用的相关接口")
            .build();

    }

}
```

效果展示:

default

抽奖系统前端api

[主页](#)

[Swagger Models](#)

[文档管理](#)

[抽奖模块](#)

[活动模块](#)

[GET 中奖列表](#)

[GET 活动信息](#)

[GET 活动列表](#)

[GET 奖品信息](#)

[测试模块](#)

[用户模块](#)

[登录模块](#)

[文档](#)

[调试](#)

GET

/api/game/hit/{gameid}/{curpage}/{limit}

发送

全选	参数类型	参数名称	参数值
<input checked="" type="checkbox"/>	path	curpage	1
<input checked="" type="checkbox"/>	path	gameid	1
<input checked="" type="checkbox"/>	path	limit	2

响应内容

Raw

Headers

Curl

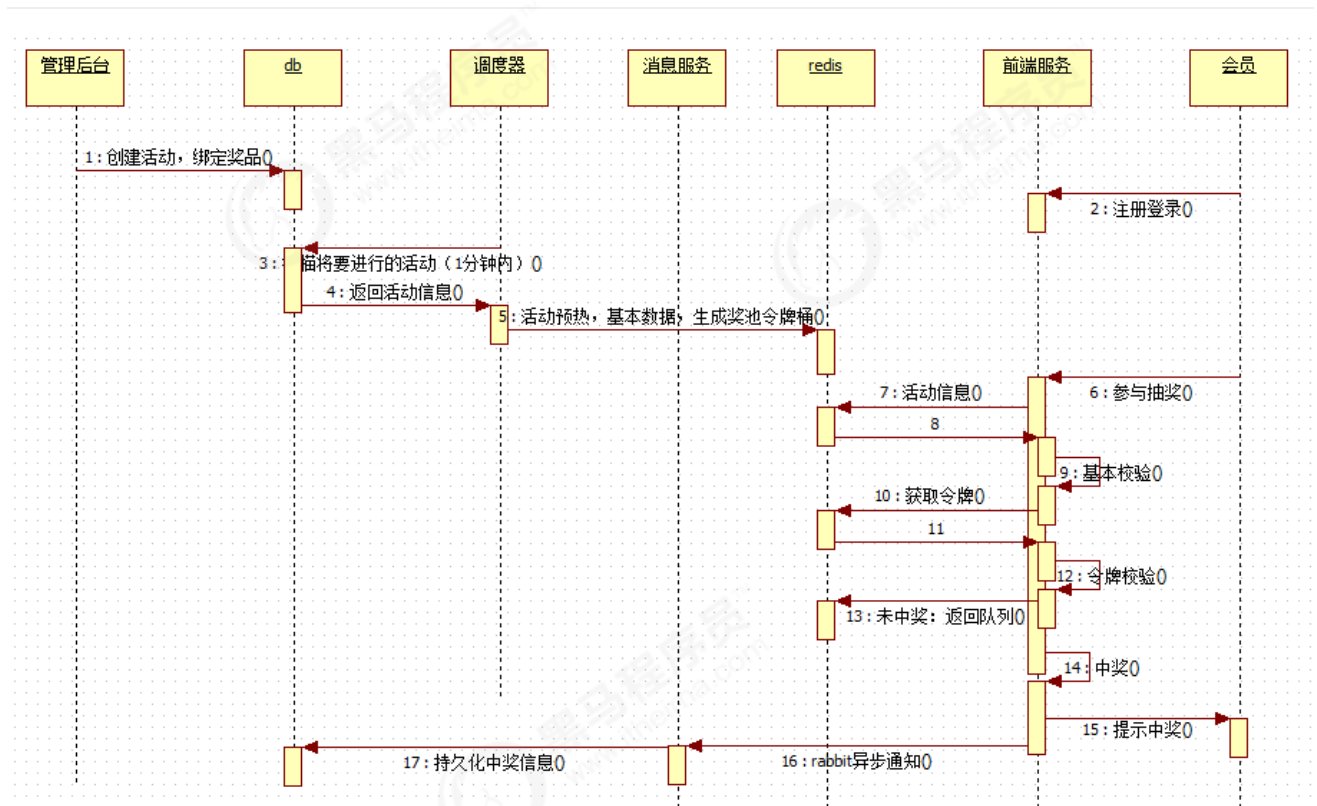
响应码:200 OK耗时:29 ms大小:568 b

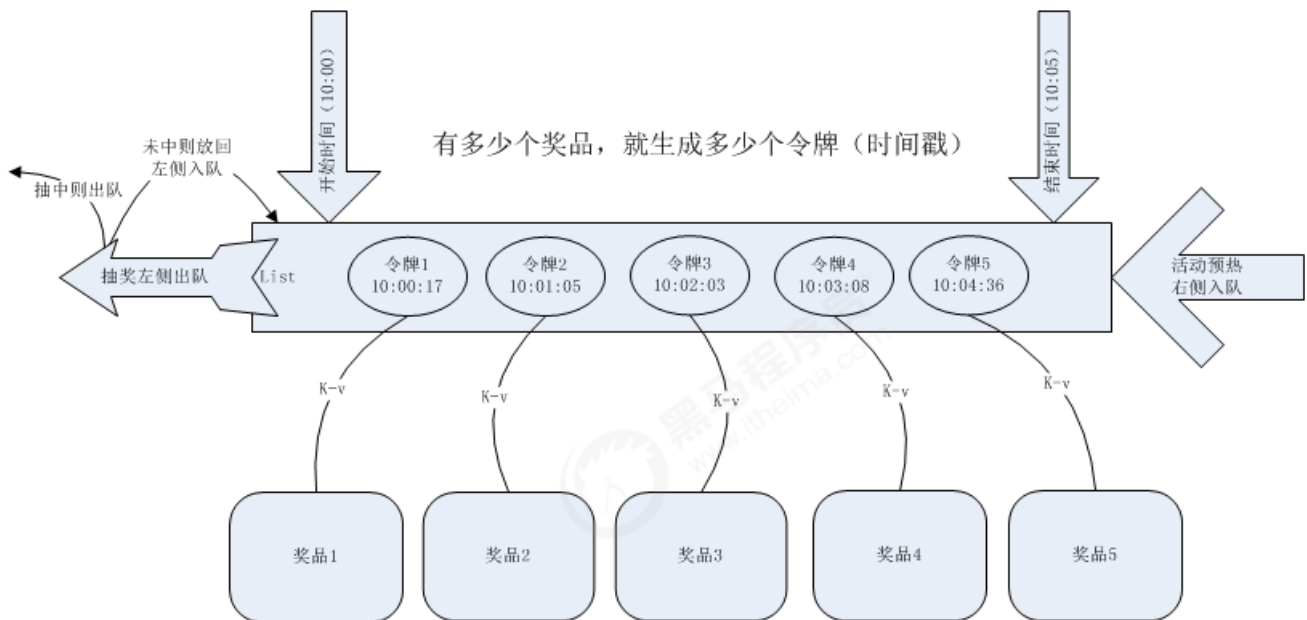
```

1 {
2   "code": 1,
3   "msg": "成功",
4   "data": {
5     "currentPage": 1,
6     "pageSize": 2,
7     "totalNum": 5,
8     "isMore": 1,
9     "totalPage": 3.

```

3.代码解读





缓存结构设计回顾：

- 活动开始前1分钟扫描将要开始的活动
- 将活动信息加载进redis
- 将活动策略信息加载进redis
- 按活动奖品信息，生成对应个数的时间戳做令牌，从小到大排好序，右侧入队
- 以令牌为key，对应的奖品为value，建立映射关系，为中奖后获取奖品做准备
- 抽奖开始时，从令牌队列左侧获取令牌
- 如果令牌小于当前时间，说明中奖，找到令牌对应的奖品，抽走
- 如果令牌大于当前时间，说明未中奖，从左侧将令牌放回队列

3.1 活动预热

3.1.1 源码实现及解读



```
package com.itheima.prize.job.task;

import ...

/**
 * 活动信息预热，每隔1分钟执行一次
 * 查找未来1分钟内（含），要开始的活动
 */
@Component
@ElasticSimpleJob(cron = "0 * * * * ?")
public class GameTask implements SimpleJob {
    private final static Logger log = LoggerFactory.getLogger(GameTask.class);
    @Autowired
    private CardGameMapper gameMapper;
    @Autowired
    private CardGameProductMapper gameProductMapper;
    @Autowired
    private CardGameRulesMapper gameRulesMapper;
    @Autowired
    private GameLoadMapper gameLoadMapper;
    @Autowired
    private RedisUtil redisUtil;

    @Override
    public void execute(ShardingContext shardingContext) {
        //当前时间
        Date now = new Date();
        //查询将来1分钟内要开始的活动
        CardGameExample example = new CardGameExample();
        CardGameExample.Criteria criteria = example.createCriteria();
        //开始时间大于当前时间
        criteria.andStarttimeGreaterThan(now);
        //小于等于（当前时间+1分钟）
        criteria.andStarttimeLessThanOrEqualTo(DateUtils.addMinutes(now,1));
        List<CardGame> list = gameMapper.selectByExample(example);
        if(list.size() == 0){
            //没有查到要开始的活动
            log.info("game list scan: size = 0");
            return;
        }
        log.info("game list scan : size = {}",list.size());
        //有相关活动数据，则将活动数据预热，进redis
        list.forEach(game ->{
            //活动开始时间
            long start = game.getStarttime().getTime();
            //活动结束时间
            long end = game.getEndtime().getTime();
            //计算活动结束时间到现在还有多少秒，作为redis key过期时间
            long expire = (end - now.getTime())/1000;
            //
            long expire = -1; //永不过期
            //活动持续时间（ms）
            long duration = end - start;
        });
    }
}
```



```
//活动基本信息
redisUtil.set(RedisKeys.INFO+game.getId(),game,-1);
log.info("load game info:{},{},{},{}",
game.getId(),game.getTitle(),game.getStarttime(),game.getEndTime());

//活动奖品信息
List<CardProduct> products = gameLoadMapper.getByGameId(game.getId());
Map<Integer,CardProduct> productMap = new HashMap<>(products.size());
products.forEach(p -> productMap.put(p.getId(),p));
log.info("load product type:{",productMap.size());

//奖品数量等配置信息
CardGameProductExample productExample = new CardGameProductExample();
productExample.createCriteria().andGameidEqualTo(game.getId());
List<CardGameProduct> gameProducts =
gameProductMapper.selectByExample(productExample);
log.info("load bind product:{",gameProducts.size());

//令牌桶
List<Long> tokenList = new ArrayList();
gameProducts.forEach(cgp ->{
    //生成amount个start到end之间的随机时间戳做令牌
    for (int i = 0; i < cgp.getAmount(); i++) {
        long rnd = start + new Random().nextInt((int)duration);
        //为什么乘1000，再额外加一个随机数呢？ - 防止时间段奖品多时重复
        //记得取令牌判断时间时，除以1000，还原真正的时间戳
        long token = rnd * 1000 + new Random().nextInt(999);
        //将令牌放入令牌桶
        tokenList.add(token);
        //以令牌做key，对应的商品为value，创建redis缓存
        log.info("token -> game : {} -> {}",token/1000
,productMap.get(cgp.getProductid()).getName());
        //token到实际奖品之间建立映射关系
        redisUtil.set(RedisKeys.TOKEN + game.getId()
+ "_" + token,productMap.get(cgp.getProductid()),expire);
    }
});
//排序后放入redis队列
Collections.sort(tokenList);
log.info("load tokens:{",tokenList);

//从右侧压入队列，从左到右，时间戳逐个增大
redisUtil.rightPushAll(RedisKeys.TOKENS + game.getId(),tokenList);
redisUtil.expire(RedisKeys.TOKENS + game.getId(),expire);

//奖品策略配置信息
CardGameRulesExample rulesExample = new CardGameRulesExample();
rulesExample.createCriteria().andGameidEqualTo(game.getId());
List<CardGameRules> rules = gameRulesMapper.selectByExample(rulesExample);
//遍历策略，存入redis hset
rules.forEach(r -> {

    redisUtil.hset(RedisKeys.MAXGOAL
```

```
+game.getId(),r.getUserLevel()+"",r.getGoalTimes());
        redisUtil.hset(RedisKeys.MAXENTER
+game.getId(),r.getUserLevel()+"",r.getEnterTimes());
        log.info("load rules:level={},enter={},goal=
{}",r.getUserLevel(),r.getEnterTimes(),r.getGoalTimes());
    });
    redisUtil.expire(RedisKeys.MAXGOAL +game.getId(),expire);
    redisUtil.expire(RedisKeys.MAXENTER +game.getId(),expire);

    //活动状态变更为已预热，禁止管理后台再随便变动
    game.setStatus(1);
    gameMapper.updateByPrimaryKey(game);
    });
}
```

3.1.2 过期时间

活动结束时间 - 当前时间 = 有效时长，活动结束则缓存自动失效

3.1.3 调度策略

每隔1分钟扫描一遍活动表，查询未来1分钟内要开始的活动进行预热。

3.2抽奖业务



```
@GetMapping("/go/{gameid}")
@ApiOperation(value = "抽奖")
@ApiImplicitParams({
    @ApiImplicitParam(name="gameid",value = "活动id",example = "1",required = true)
})
public ApiResult<Object> act(@PathVariable int gameid, HttpServletRequest request){
    Date now = new Date();
    //获取活动基本信息
    CardGame game = (CardGame) redisUtil.get(RedisKeys.INFO+gameid);
    //判断活动是否开始
    //如果活动信息还没加载进redis, 无效
    //如果活动已经加载, 预热完成, 但是开始时间 > 当前时间, 也无效
    if (game == null || game.getStarttime().after(now)){
        return new ApiResult(-1,"活动未开始",null);
    }
    //判断活动是否已结束
    if (now.after(game.getEndtime())){
        return new ApiResult(-1,"活动已结束",null);
    }
    //获取当前用户
    HttpSession session = request.getSession();
    CardUser user = (CardUser) redisUtil.get(RedisKeys.SESSIONID+session.getId());
    if (user == null){
        return new ApiResult(-1,"未登陆",null);
    }
    //用户已中奖次数
    Integer count = (Integer) redisUtil.get(RedisKeys.USERHIT+gameid+"_"+user.getId());
    if (count == null){
        count = 0;
        redisUtil.set(RedisKeys.USERHIT+gameid+"_"+user.getId(),count,
            (game.getEndtime().getTime() - now.getTime())/1000);
    }
    //根据会员等级, 获取本活动允许的最大中奖数
    Integer maxcount = (Integer)
redisUtil.hget(RedisKeys.MAXGOAL+gameid,user.getLevel()+"");
    //如果没设置, 默认为0, 即: 不限制次数
    maxcount = maxcount==null ? 0 : maxcount;
    //次数对比
    if (maxcount > 0 && count >= maxcount){
        //如果达到最大次数, 不允许抽奖
        return new ApiResult(-1,"您已达到最大中奖数",null);
    }

    //以上校验全部通过, 准许进入抽奖逻辑
    Long token = (Long) redisUtil.leftPop(RedisKeys.TOKENS+gameid);
    if (token == null){
        //令牌已用光, 说明奖品抽光了
        return new ApiResult(-1,"奖品已抽光",null);
    }
    //判断令牌时间戳大小, 即是否中奖
    //这里记住, 取出的令牌要除以1000, 参考job项目, 令牌生成部分
    if (now.getTime() < token/1000){
        //当前时间小于令牌时间戳, 说明奖品未到发放时间点, 放回令牌, 返回未中奖
    }
}
```



```
        redisUtil.leftPush(RedisKeys.TOKENS+gameid,token);
        return new ApiResult(0,"未中奖",null);
    }

    //以上逻辑走完，说明很幸运，中奖了！
    //抽中的奖品：
    CardProduct product = (CardProduct) redisUtil.get(RedisKeys.TOKEN + gameid + "_" + token);
    //中奖次数加1
    redisUtil.incr(RedisKeys.USERHIT+gameid+"_"+user.getId(),1);
    //投放消息给队列，中奖后的耗时业务，交给消息模块处理
    Map map = new HashMap(2);
    map.put("gameid",gameid);
    map.put("userid",user.getId());
    map.put("productid",product.getId());
    map.put("hittime",now.getTime());
    rabbitTemplate.convertAndSend(RabbitKeys.EXCHANGE_DIRECT,RabbitKeys.QUEUE_HIT, map);
    //返回给前台中奖信息
    return new ApiResult(1,"恭喜中奖",product);
}
```

并发问题：lua脚本处理

```
local token = redis.call('lpop',KEYS[1])
local curtime = tonumber(KEYS[2])

if token ~= false then
    if ( tonumber(token)/1000 > tonumber(KEYS[2]) ) then
        redis.call('lpush',KEYS[1],token)
        return 1
    else
        return tonumber(token)
    end
else
    return 0
end
```

调用部分：

```
Long token = luaScript.tokenCheck("game_"+gameid,String.valueOf(new Date().getTime()));
```

3.3 中奖处理

3.3.1 rabbit配置



```
package com.itheima.prize.commons.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.amqp.core.*;
import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
import org.springframework.amqp.rabbit.connection.CorrelationData;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * rabbit配置
 */
@Configuration
public class RabbitConfig {
    private static Logger log = LoggerFactory.getLogger(RabbitConfig.class);
    @Autowired
    private CachingConnectionFactory connectionFactory;

    // 队列
    @Bean
    public Queue getQueue1() {
        return new Queue(RabbitKeys.QUEUE_HIT);
    }

    // 路由
    @Bean
    DirectExchange directExchange() {
        return new DirectExchange(RabbitKeys.EXCHANGE_DIRECT);
    }

    // 绑定队列于路由
    @Bean
    Binding bindingExchangeDirect() {
        return BindingBuilder.bind(getQueue1())
            .to(directExchange()).with(RabbitKeys.QUEUE_HIT);
    }

    @Bean
    public RabbitTemplate rabbitTemplate() {
        ...
        return rabbitTemplate;
    }
}
```

3.3.2 异步处理

中奖主流程中，将中奖信息放入消息队列，立刻返回结果响应前台。

//投放消息给队列，中奖后的耗时业务，交给消息模块处理

```
Map map = new HashMap(2);
map.put("gameid",gameid);
map.put("userid",user.getId());
map.put("productid",product.getId());
map.put("hittime",now.getTime());
rabbitTemplate.convertAndSend(RabbitKeys.EXCHANGE_DIRECT,RabbitKeys.QUEUE_HIT, map);
```

3.3.3 msg模块消费

```
@RabbitHandler
public void processMessage3(Map message) {
    logger.info("user hit : " + message);
    CardUserHit hit = new CardUserHit();
    hit.setGameid(MapUtils.getIntValue(message,"gameid"));
    hit.setUserId(MapUtils.getIntValue(message,"userid"));
    hit.setProductid(MapUtils.getIntValue(message,"productid"));
    hit.setHittime(new Date(MapUtils.getLongValue(message,"hittime")));
    // 入库
    hitMapper.insert(hit);
}
```

3.4 缓存监控

3.4.1 代码展示

```
@GetMapping("/info/{gameid}")
@ApiOperation(value = "缓存信息")
@ApiImplicitParams({
    @ApiImplicitParam(name="gameid",value = "活动id",example = "1",required = true)
})
public ApiResult info(@PathVariable int gameid){
    Map map = new LinkedHashMap<>();
    map.put(RedisKeys.INFO+gameid,redisUtil.get(RedisKeys.INFO+gameid));
    List<Object> tokens = redisUtil.lrange(RedisKeys.TOKENS+gameid,0,-1);
    Map tokenMap =new LinkedHashMap();
    tokens.forEach(o -> tokenMap.put(
        new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS")
            .format(new Date(Long.valueOf(o.toString())/1000)),
        redisUtil.get(RedisKeys.TOKEN + gameid + "_" + o)
    ));
    map.put(RedisKeys.TOKENS+gameid,tokenMap);
    map.put(RedisKeys.MAXGOAL+gameid,redisUtil.hmget(RedisKeys.MAXGOAL+gameid));
    map.put(RedisKeys.MAXENTER+gameid,redisUtil.hmget(RedisKeys.MAXENTER+gameid));
    return new ApiResult(200,"缓存信息",map);
}
```

3.4.2 结果展示



```
{
  "code": 200,
  "msg": "缓存信息",
  "data": {
    "game_info_5": {
      "id": 5,
      "title": "财务自由就靠这了",
      "info": "财务自由就靠这了财务自由就靠这了财务自由就靠这了财务自由就靠这了财务自由就靠这了财务自由就靠这了财务自由就靠这了财务自由就靠这了财务自由就靠这了",
      "starttime": "2019-10-08 10:21:46",
      "endtime": "2019-10-08 16:43:46",
      "type": 2,
      "status": 1
    },
    "game_tokens_5": {
      "2019-10-08 10:34:07.662": {
        "id": 8,
        "name": "不长胖奶茶一杯",
        "pic": "/upload/images/QQ图片20190903143639.jpg",
        "info": "奶茶原为中国北方游牧民族的日常饮品，至今最少已有千年历史。自元朝起传遍世界各地，目前在大中华地区，中亚国家，印度，阿拉伯，英国，马来西亚，新加坡等地区都有不同种类奶茶流行。蒙古高原和中亚地区的奶茶千百年来从未改变，至今仍然是日常饮用及待客的必备饮料。",
        "price": 50
      },
      "2019-10-08 10:39:23.571": {
        "id": 5,
        "name": "美国商务办公室一套",
        "pic": "/upload/images/123d.jpg",
        "info": "白宫（英语：The White House）也称为白屋，是美国总统的官邸和办公室。1902年被西奥多·罗斯福总统正式命名为“白宫”。白宫由美国国家公园管理局拥有",
        "price": 1000
      },
      "2019-10-08 10:41:51.542": {
        "id": 8,
        "name": "不长胖奶茶一杯",
        "pic": "/upload/images/QQ图片20190903143639.jpg",
        "info": "奶茶原为中国北方游牧民族的日常饮品，至今最少已有千年历史。自元朝起传遍世界各地，目前在大中华地区，中亚国家，印度，阿拉伯，英国，马来西亚，新加坡等地区都有不同种类奶茶流行。蒙古高原和中亚地区的奶茶千百年来从未改变，至今仍然是日常饮用及待客的必备饮料。",
        "price": 50
      },
      "2019-10-08 10:46:43.439": {
        "id": 1,
        "name": "iPhoneX一部",
        "pic": "/upload/images/aaa123.JPEG",
        "info": "iPhone X是美国Apple（苹果公司）于北京时间2017年9月13日凌晨1点，在Apple Park新总部的史蒂夫·乔布斯剧院会上发布的新机型。其中“X”是罗马数字“10”的意思，代表着苹果向iPhone问世十周年致敬。",
        "price": 5000
      },
      "2019-10-08 10:52:53.002": {
        "id": 6,
        "name": "地下车位一个",
        "pic": "/upload/images/asdasd2.jpg",
```

"info": "地库设计导则 - 说明 ? 本指引针对的项目类型仅为区域内GCT主流项目。 ? 本指引是在广深区域目前操作项目的前提下,根据各项目经验梳理出的小结.",

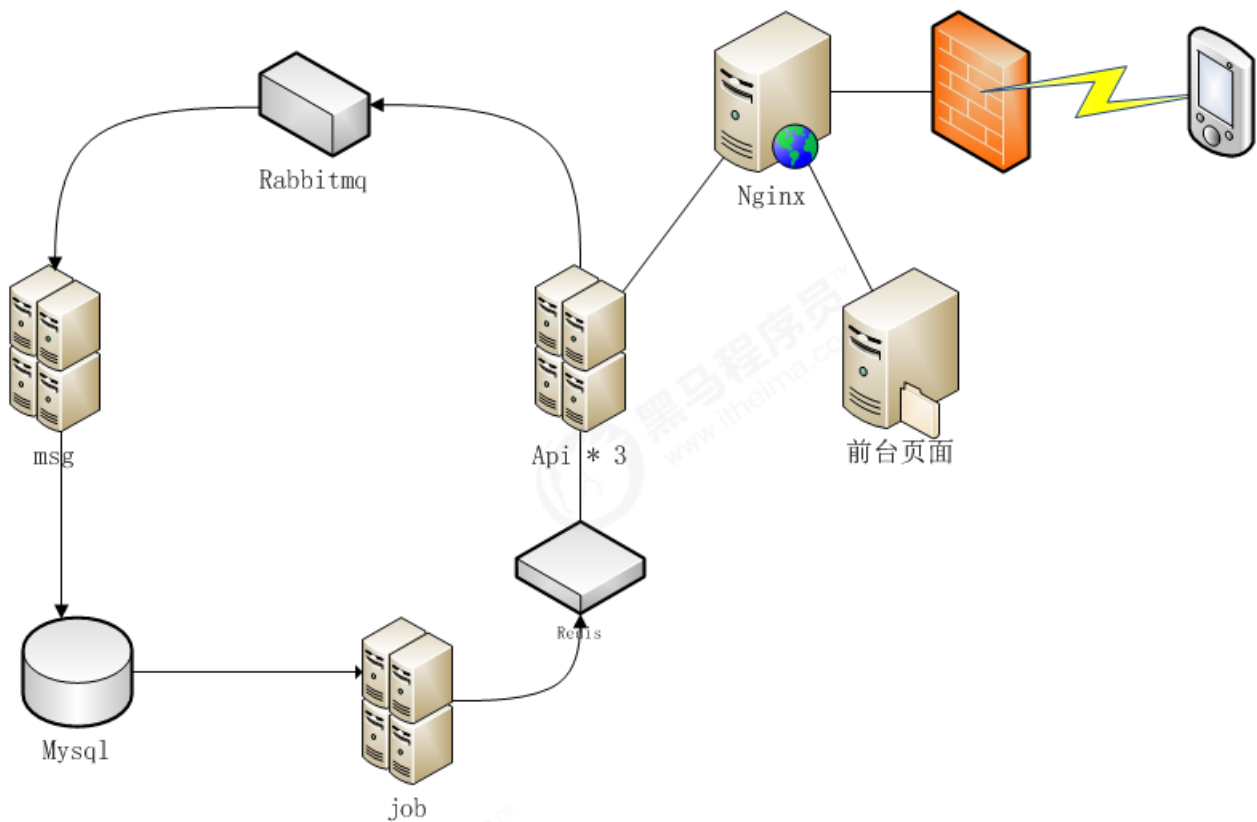
```
"price": 500
},
....
},
"game_maxgoal_5": {
  "0": 1,
  "1": 2,
  "2": 3,
  "3": 4,
  "4": 0
},
"game_maxenter_5": {
  "0": 0,
  "1": 0,
  "2": 0,
  "3": 0,
  "4": 0
}
}
}
```

3.5代码启动

- 启动中间件: mysql, redis, rabbitmq, zookeeper
- 启动Eureka
- 启动job
- 启动msg
- 启动api
- 启动管理后台, 创建活动
- 查看job日志, 是否在开始前的1分钟, 将活动预热成功
- 请求抽奖接口, 查看是否提示活动未开始
- 开始时间后, 请求抽奖接口, 调试api, 看是否可以正常抽奖
- 中奖后, 看msg日志, 是否有中奖消息发送到消息模块
- 查看数据库, 确认中奖信息是否正常写入db

3.6上线部署

3.6.1 部署拓扑



3.6.2 jenkins集成

1) 任务汇总:

<div> build install 所有 + </div>			
S	W	名称 ↓	上次成功
		all-build	4 小时 27 分 - #8
		api-build	4 小时 27 分 - lastVersion=10
		api-install	4 小时 25 分 - version=10
		eureka-build	4 小时 27 分 - lastVersion=23
		eureka-install	4 小时 25 分 - version=23
		job-build	4 小时 27 分 - lastVersion=12
		job-install	4 小时 25 分 - version=12
		msg-build	4 小时 27 分 - lastVersion=10
		msg-install	4 小时 25 分 - version=10

2) 说明

每个服务，以eureka为例，分为两个阶段：build，install

build即编译阶段，从git获取代码，maven打包出springboot-jar，启用docker打成镜像，并以当前build_num为版本号。

install即部署阶段，将上一步中打出的镜像，通过docker swarm部署启动。

3) 脚本

build.sh: /opt/scripts/build.sh 服务名

```
app=$1
mv /opt/app/$app/ROOT.jar /opt/app/$app/`date +%Y%m%d-%H%M%S`.jar
cp frontend/$app/target/$app-0.0.1-SNAPSHOT.jar /opt/app/$app/ROOT.jar
cat /opt/app/Dockerfile > /opt/app/$app/Dockerfile
docker build -t $app:$BUILD_NUMBER /opt/app/$app
```

Dockerfile: jenkins脚本build阶段需要的基础镜像及模板。springboot微服务镜像全部采用该模板打包生成

```
FROM daocloud.io/library/java:openjdk-8u40-jdk
#语言字符，解决乱码问题
ENV LC_ALL C.UTF-8
ENV LANG C.UTF-8
ENV LANGUAGE C.UTF-8
#时区及时间，不设置会影响活动的开始结束
ENV TZ=Asia/Shanghai
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
#将springboot的jar包打入镜像
COPY ROOT.jar /opt/ROOT.jar
WORKDIR /opt
#启动命令，注意环境配置 prod
CMD ["java", "-jar", "ROOT.jar", "--spring.profiles.active=prod"]
```

install.sh: /opt/scripts/install.sh 服务名 版本号 端口号

```
app=$1
version=$2
port=$3

docker service rm $app
docker service create --name $app -p $port:8080 --hostname localhost $app:$version
```

3.6.3 中间件部署

zookeeper, redis, mysql, nginx 均采用docker方式部署

#获取镜像

```
docker pull zookeeper:3.4.13
docker pull daocloud.io/library/redis:latest
docker pull daocloud.io/library/rabbitmq:3.6.10-management
docker pull daocloud.io/mysql:5.7.4
docker pull daocloud.io/library/nginx:1.16.0-alpine-perl
```

#启动镜像

```
docker run --name redis -p 6379:6379 -d daocloud.io/library/redis

docker run --name mysql -v /opt/data/mysql:/var/lib/mysql -p3306:3306 -e
MYSQL_ROOT_PASSWORD=root -d daocloud.io/mysql:5.7.4

docker run -d --hostname my-rabbit --name rabbit -p 15672:15672 -p 5672:5672
daocloud.io/library/rabbitmq:3.6.10-management

docker run --name nginx -v /opt/data/nginx/html:/usr/share/nginx/html:ro -v
/opt/app/back/upload:/usr/share/nginx/upload:ro -v
/opt/data/nginx/nginx.conf:/etc/nginx/nginx.conf:ro -p 80:80 -d daocloud.io/nginx

docker run --name zookeeper -v /opt/data/zksingle:/data -p 2181:2181 -e
ZOO_LOG4J_PROP="INFO,ROLLINGFILE" -d zookeeper:3.4.13
```

3.6.4弹性扩容

采用微服务架构的初衷之一就是实现服务单元的快速独立扩容，借助docker-swarm的功能，变得很容易实现。以本项目为例，api模块在部署完成后，会自动扩容到3个实例，以负载前端的并发请求：

```
docker service scale api=3
```

```
[root@iZ8vb3a9qxfwannyw16zZ app]# docker service ls
ID                NAME      MODE                REPLICAS            IMAGE                PORTS
v71cn5e1mny4     api       replicated          3/3                 api:10              *:9004->8080/tcp
ihzcd9oiigvj     eureka   replicated          1/1                 eureka:23           *:9001->8080/tcp
p5xme0uf9czb     job       replicated          1/1                 job:12              *:9002->8080/tcp
6zyc6nbu9mcn     msg       replicated          1/1                 msg:10              *:9003->8080/tcp
```

```
[root@iZ8vb3a9qxfwannyw16zZ app]# docker service ps api
ID                NAME      IMAGE                NODE                DESIRED STATE
k880w0s5bn7z     api.1     api:10              iZ8vb3a9qxfwannyw16zZ Running
nw8kejxhvkqk     api.2     api:10              iZ8vb3a9qxfwannyw16zZ Running
q32v40vpkxso     api.3     api:10              iZ8vb3a9qxfwannyw16zZ Running
```

3.6.5 管理后台部署

因为是开发平台，打出的war包，这里使用jetty启动即可

```
java -jar /opt/app/jetty-runner.jar --port 7070 /opt/app/back/
```

3.6.6 静态页面与nginx

/api的请求代理到微服务api模块，其他静态文件由nginx本地访问，配置如下：

```
http {  
  
    include      mime.types;  
    default_type application/octet-stream;  
    sendfile      on;  
    keepalive_timeout 65;  
    server {  
        listen      80;  
        server_name localhost;  
        location ^~ /api/ {  
            proxy_pass http://172.17.0.1:9004;  
        }  
        location ^~ /upload/ {  
            alias /usr/share/nginx/upload/;  
        }  
        location / {  
            root /usr/share/nginx/html;  
        }  
    }  
}
```

3.7 发散思维与总结

3.7.1 发散思维

1) lua脚本的运用

使用lua脚本，将抽奖的逻辑从java端移入redis服务器端，作为一个整体函数暴露给java调用，减少了java服务器与redis服务器之间的通信次数，性能会得到提升。

2) 怎么实现活动暂停功能？

要实现活动随时暂停，可以新增一个接口，该接口修改redis缓存中的活动状态。

抽奖接口逻辑中增加暂停状态判断。如果是暂停，返回给前台以提示。

3) 怎么实现多种投放策略？

投放策略即令牌的生成策略不同。可以修改令牌生成部分代码。按递增，递减，正态分布等多种函数生成时间戳。

3.7.2 总结

通过本项目我们接触到了以下知识点：

- 中间件的运用，zookeeper，rabbitmq，尤其是redis的数据结构
- springcloud框架的搭建与集成
- 一些工具：分页PageHelper，swagger2，mybatis-generator，zkui
- 微服务思想及持续集成，动态扩容
- 基于docker swarm的发布与部署