关于广告领取并发问题解决方案

1. 项目背景：

广告商可以发布，积分广告，现金广告，现金和积分广告，答题广告，广告通过系统审核上线后在在，广告便可以在具体的时间在view视图中展示出来，答题广告需求答题完毕才能获取广告商设置的领取奖赏，领取完后，用户账户的账户信息便会到账。广告领取剩余一定量的时候，要求通知广告商广告账户余额不足。

1. 流程分析：
   1. 广告从创建到销毁的整个生命周期来看。
      1. 创建广告---》 审核广告----》审核通过----》 上线展示》------》下线。 等阶段。
2. 广告赠送
   1. 广告赠送会做什么：
      1. 判断广告是否上线，是否余额充足。
      2. 有的话，用户可以领取，没有的话，提示相关信息。
      3. 领取的逻辑是，广告账户里扣钱，用户账户加钱，（一起失败一起成功）。
3. 分析：阻碍广告并发领取的瓶颈是：
   1. 数据库读写： 数据库读写数据IO操作。会花费大量的时间。
   2. 业务处理速度速度瓶颈： 每次用户请求的时候都会涉及大量对象创建销毁。浪费系统资源。
4. 浅谈缓存容器：

java 开发能用到的数据存储，操作容器大致分为以下几类：

* + - 1. Jvm 线程栈缓存： 该缓存只在程序动态运行时缓存基本类型数据。
      2. Jvm 内存堆： jvm 内存堆物理内存是机器主内存，缓存java对象。
      3. 普通容器：map set list 等，普通容器中的实现是利用算法，实现数据存取，普通容器中的大部分类工具，只能在单线程中高性能运行，是java

语言大厦的基石。

* + - 1. 并发容器：java普通中容器中，虽然提供了部分的线程安全的类，单其底层实现大多都是java锁，导致效率低下，所以针对并发与线程安全的数据存储，java1.5 中，终于推出了并发容器，其实现也是利用算法，重构原有普通容器的各种数据结构，细化数据操作的锁粒度。

如ConcurrentHashMap

1.8之前的原本实现是：Segment 数组hash成table然后给table，再在table上挂链表，在Segment数组上加锁，用了一种降锁粒度的方式来实现，而1.8 利用CAS 算法+node红黑数+同步手段来实现。

5. 常用的数据存储和数据加工工具：

Redis： redis 是高性能分布式数据库，能给分布式系统提供强大的数据存储与原子性运算支持，它是一种内存数据库。不足之处就是分布式是socket api 导致性能降低，但比关系型数据库性能要高的多。

Msql- oracle -sqlserver： 关系型数据库，是大型程序的数据基本库。

Ehcache： 他也是内存数据库，与redis 数据库相比，性能肯定比redis高，但是有缓存共享瓶颈，集群应用不太方便，单机应用，有很大的优势。

好了，话不多说：现在来看看我的具体设计与解决：

解决方案与技术选型：

在并发速度与业务处理速度，IO速度不匹配的情况下，我的设计是缓存-异步处理机制来减缓IO阻塞时间，将及时响应与耗时处理解耦。达到优化系统性能的目的。

前置工作：

* + - 1. 安装 redis 数据库
         1. 去<http://www.redis.cn/> 安装一个redis 数据库。就像安装其他数据库一样。安装完成之后运行控制台命令行运行redis-server ，如配置了配置文件，如用户验证，等其他信息，运行时需要在安装目录下

运行redis-server XXX.conf ，如window，安装，环境变量默认为redis-server配置了，如文件找不到或错误，需要定位到安装目录运行，或者给配置文件也配置个环境变量。

b） 服务关闭时，存储的数据将被清空，程序可以通过参数将数据持久化 ，但是需要一定的性能代价。

* + - 1. 目前，项目用的是jfinal微框架，其框架已经实现了redis 相关操作接口，
         1. 第一步，是在jfinal中配置类的Plugins 容器中配置上他们提供的redisPlugin类的实例。配置正确后，方可链接上redisserver
         2. Jfnal 提供了对Redis 数据库操作，通过redis.类可实现对数据的基本操作，通过redis.use(“”),可切换数据源。

注意：redis 操作写入,其底层是通过byte数组写入的,这就以为这你可以缓存任何数据.只要不超过redis缓存容量限制. Jfinal提供的redis 写入有两种模式,一种是明文模式,一种是加密模式,两种模式的实现,是依赖于实现了对象的序列化器. FstSerializer 和JDKSerializer 序列化器,写入和读取,必须使用同一个序列化器,否则报错.

方案：

I: 初始化处理：

首先，我们为什么要初始时处理，就好比说你买了个跑车，但是在泥巴路上，就算你车再牛，也肯定没有在好路上的车跑的快。那么，并发的初始工作我们要做些什么呢？

我们的思想是利用高性能的工具，redis，更或者是其他高性能工具，首先，在应用开始服务之前，我们需要将传统数据库数据的核心数据放置到高性能工具上，但是业务上要注意缓存与数据库的数据一致性。

这里我们的核心工作：

将上线广告核心数据写入缓存，并发的数据，我们将不再处理，虽然也能处理，但是web 应用或者分布式系统中，对底层要求不是很高的，与业务有关的数据，可以采用这种高性能工具来实现，如redis 提供了原子性增量接口，很方便。

* + 1. 前置处理：
       1. 前置处理的核心工作:
          1. 过滤用户请求： 请求由页面端传递，需要高速的过滤掉无效请求，和非法请求。

无效请求： 用户点按钮，有可能短时间内多次触发按钮请求后台，此时我们将用cache命中与命不中来过滤掉请求，使得合法请求等通过，不合法请求将被拦截，直接返回给用户。及时腾出系统资源。

* + - 1. 并将请求用户入队。
      2. 缓存控制。当请求达到某个阀值是，我们需要操控缓存来控制业务是否结束或者做其他操作。
    1. 后置处理解决方案：

后置异步处理：

后置异步处理，是基于JMS消息模型概念，变相来实现系统的负责均衡，有处理系统根据自己的处理能力自行拉去处理任务。

消息中间件的选择方案：

消息中间件的选择要依据整个系统的规划与布局来选择消息中间件，

消息中间有哪些：

市面上常见的RabbitMQ， activeMQ，Kafa 等。

这些消息中间件都是常见的工具，使用方法，其性能，优劣势也是各有千秋。

Redis： redis有持久化功能，元素弾栈 等功能，也可应用于消息队列，但其业务领域并非专业消息中间件。

Redis与专业消息中间的对比：

显然redis 的高性能，高并发优势，是这些消息中间件无人能及的，但是需要我们程序员来自己实现其消息的架构，实现消息管理等操作，如系统非常复杂，实现起来也是非常复杂的。其应用场景是，小规模并发，小规模集群的架构，可选。其他专业消息中间件，其性能没有redis好，但是其有着强大的路由功能，能容易实现大规模集群，与消息管理。有着比redis更高的可靠性。

* + - 1. 第一种方案：然后将数据封装为消息对象，扔到消息消息队列。将消息队列采用生产异步消费者模式。来处理业务。这种方案可易于实现分布式处理
      2. 第二种方案： 还是采用redis的cache来做消息队列，后置处理器自定义多线程容器，来处理。
    1. 详细方案：
       1. 前置控制处理器：
          1. 我们利用redis数据库操作单个操作的原子性，来实现前置并发。

首先，我们将cache分为三层：

第一层cache ：

view cache :该cache是为面向用户的cache，只有命中用户，便可进入下一层cache。

cache设计及内容：本层cache采用set数据结构将AD对象cache名称存入该cache 。

第二层cache：

limit cache :该cache是对用户请求做限制判断与处理，（如广告领取用户在一秒钟内点击了几次，我们要限制其请求）对命中第一层cache的用户来实现处理，通过该cache的用户将提交到后置处理器。

第二层cache：

count cache :该cache是对并发前置处理，对命中第一层cache的用户来实现处理，通过该cache的用户将提交到后置处理器。

cache设计及内容: 该cache采用set数据结构。Kvalue为广告的并发次数。

利用redis 自减操作原子性并发控制用户请求。

第三层 cache：

core cache ：核心cache 存储AD cache 信息， 为后置处理器提供最重要的数据支持。

内容：该cache采用hashset数据结构。Key为AD cache名称：value为广告的并发次数。

本层cache意义：

由于采用异步处理机制。当AD下线时退款时，第一层，第二层cache会被立即清除，但由于队列中存在合法用户，当合法用户请求处理完时才清除该cache。

* + - 1. 后置处理器：

对消息的异步处理：

前面我们也说过，后置处理是主动负载，自己拉去任务，进行处理，如我们使用rabbitMQ，市面上已经有了相关的客户端容器来主动拉去任务，如spring-data-mqjq，但是由于不确定自己是使用redis，自实现消息队列，还是用rabbitMQ ，所以我自己实现了后置服务处理器。

1. 使用RabbitMq需要安装rebbitMQ 服务器 http://www.rabbitmq.com/
2. 由于目前jfinal无对RabbitMQ的集成那么就要自己实现jfinal框架的RabbitMQ集成。
   1. Jfnal是可对外扩展的插件式框架。要引用第三方的功能模块需要实现IPlugin 接口，来实现接口接入。
   2. RabbitMQ数据源和其他数据库数据源相似，需要从ConnectionFactory获取一个连接，然后打开虚拟信道发送消息。其连接Connection 对象数受制于配置文件，和RabbitMQserver限制，虚拟信道channel 理论上可以无穷的创建，通过channel 传输的数据，但是底层是Connection 对象采用队列算法，所以channel并不是越多越好，也需要我们使用池技术来维护虚拟信道。
   3. 抽象消息队列：RabbitMQ没有提供队列的抽象类让我们继承， 我们需要将队列通用属性进行抽象与封装RabbitQueue。

private boolean durable;// 是否持久到硬盘

private boolean exclusive;// 独占

private boolean autoDelete;// 自动删除

private Map<String, Object> arguments; //其他参数

private String queueName;//队列名称

private T t;数据对象

1. 由于是通过channel 对象来传输数据，我们需要实现自己的序列化与反序列化器，这里我自己利用JDK 实现了序列化与反序列化器。JDKSerialize
2. 发送消息： 发送消息需要调用底层的channel 和自己抽象的RabbitQueue 和序列化器功能来实现消息的推送。默认情况下消息发送将通过默认链接的虚拟信道进行消息推送，如需要发送到不同的RabbitMQ 服务器，需要传入信道channel 对象。将这些信息封装成一个MessageSender。
3. 消息接受： 和MessageSender 相反，接收消息，需要RabbitQueue信息和反序列化，信道channel 对象，和接收消息的策略。如服务主动接收，通过事件触发获取。目前暂时未实现MessageSender

后置服务写法：

服务协议： 一般要实现 init start destroy ，service 等方法

目前我们的项目框架jfinal提供 afterStart 方法，beforestop 方法 中调用我们的服务实例方法，来启动这个服务容器。

注意： jfinal是由tomca线程启动，其框架并不是一个线程容器，阻塞器tomcat线程要报服务超时异常。所以我们在调用stat方法后为自己的服务开启一个容器。

由于是主动拉去任务：

我们需要有一个监听线程。

！！其他：

关于lombook插件： 源码级编译插件工具通过注解简化代码的编写

作用:简化[Logger](eclipse-javadoc:%E2%98%82=ConcurrentDemo/src%5C/main%5C/java%3CJfinalConfig%7BJfinalConfig.java%E2%98%83JfinalConfig%5Elog%E2%98%82org.slf4j.Logger) 和loggerFactory的对象引用。和Bean对象的get set方法的省略编写。 等。

使用方法：

1. lombook 插件在项目中需要导入

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.16.20</version>

</dependency>

1. ide支持，需要在eclipse 的配置文件



加入配置信息： -javaagent:F:\EclipseForWeb\newEclipse\eclipse\lombok.jar