* 背景：在618的节假日下写一个秒杀系统的文档，实现高并发，高读写以及可以随时撤离的秒杀组件
* 目标：秒杀系统的添加是为了系统的多样性，给用户带来的福利可以留住一些常用用户。 添加的组件有购物秒杀和抢红包秒杀
* 当前解决方案：利用队列的排序机制，挑选出率先加入队列的用户，并以此用户当作成员；用户的页面分为两种页面，静态和动态，将大部分没有改变的页面显示改为静态响应，只将小部分内容设置为动态响应，减少系统的响应复杂程度；利用非关系型数据库的响应速度，为动态页面添加内容，加快页面刷新，访问是的响应速度。
* 团队配合方面：增加前端和研发团队的负担；需要增设秒杀系统服务器；

但秒杀系统的流量过大时，可能给交易组件和购物组件造成负担，大量没有被添加到缓存的访问可能造成缓存穿透，一部分缓存突然的失效可能造成缓存雪崩。  
客服团队应该想好当造成系统崩溃时，对外的话术

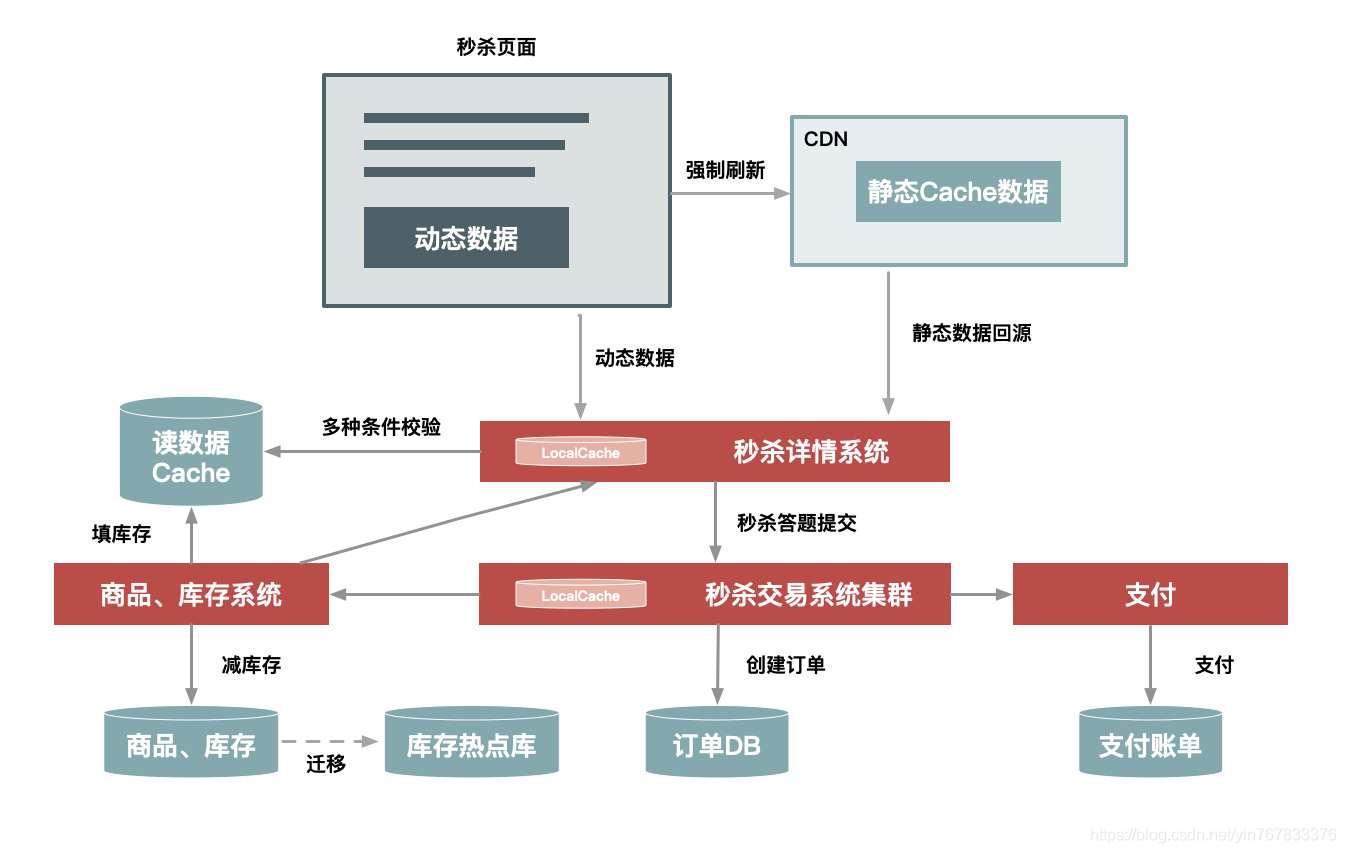
**实施细节**

1）对页面进行彻底的动静分离，使得用户秒杀时，不需要刷新整个页面，只需要点击抢宝按钮，借此把页面刷新的数据降到最少。

2）在服务端对秒杀商品进行本地缓存，不需要再调用依赖系统的后台服务获取数据，甚至不需要去公共的缓存集群中查询数据，这样不仅可以减少系统调用，而且能够避免压垮公共缓存集群。

3）增加系统限流保护，防止最坏情况发生。

经过这些优化，系统架构变成了下图中的样子。在这里，我们对页面进行了进一步的静态化，秒杀过程中不需要刷新整个页面，而只需要向服务端请求很少的动态数据。而且，最关键的详情和交易系统都增加了本地缓存，来提前缓存秒杀商品的信息，热点数据库也做了独立部署等。



4.如何做好动静分离

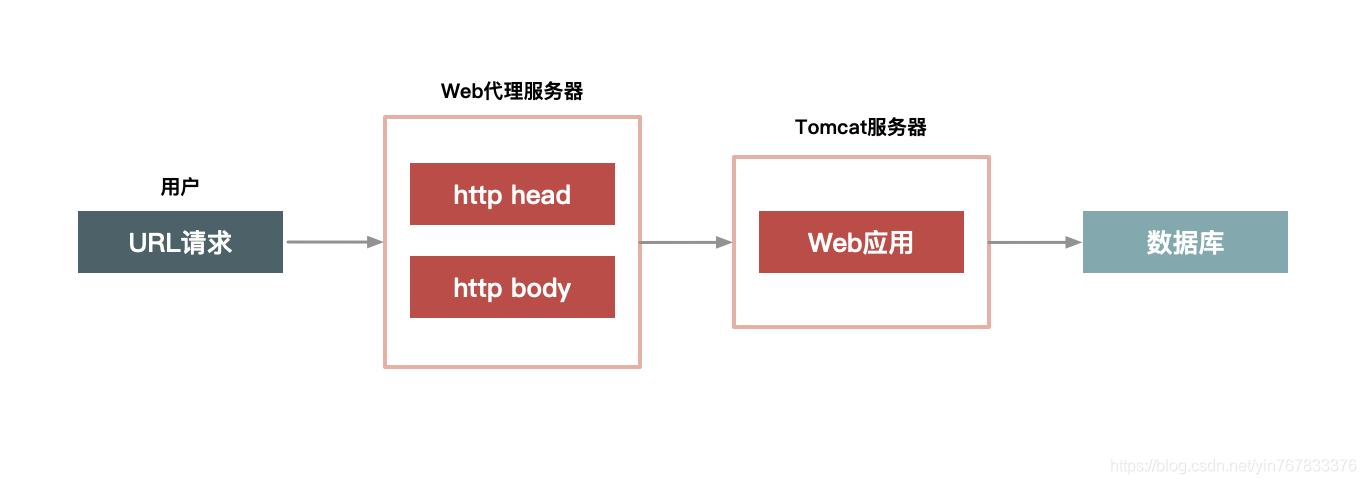
1）什么是动静分离?

动静分离：其实就是把请求的数据分为动态数据和静态数据。动态数据和静态数据的主要区别是看页面中输入的数据是否和URL、浏览者、时间、地域相关，以及是否含有 Cookie 等私密数据。

2）怎么对静态数据做缓存

第一，应该把静态数据缓存到离用户最近的地方。常见的有：缓存在用户浏览器上、CDN、服务端的Cache中。

第二，静态化改造就是要直接缓存HTTP连接。如下图所示，Web 代理服务器根据请求 URL，直接取出对应的 HTTP 响应头和响应体然后直接返回，这个响应过程简单得连 HTTP 协议都不用重新组装，甚至连 HTTP 请求头也不需要解析。



第三，让谁来缓存静态数据。不同语言写的 Cache 软件处理缓存数据的效率也各不相同。

3）如何做动静分离的改造

以典型的商品详情为例介绍，看看这个页面里都有哪些动静数据。我们从以下 5 个方面来分离出动态内容。

1.URL唯一化，商品详情系统天然地就可以做到 URL 唯一化，比如每个商品都由 ID 来标识。

2.分离浏览者相关的因素。浏览者相关的因素包括是否已登录，以及登录身份等，这些因素可以单独拆分出来，通过动态请求获取。

3.分离时间因素，服务端输出的时间通过动态请求获取。

4.异步化地域因素，页面上与地域有关的因素通过异步方式获取。

5.去掉cookie，这里说的去掉cookie并不是用户端收到的页面不含cookie了，而是缓存的静态数据中不含cookie。

4）动态内容处理方式：ESI(Edge Side Includes)方案和CSI(Client Side Include)方案

ESI(Edge Side Includes)方案：在WEB代理服务器上做动态内容请求，并将请求插入到静态页面中，当用户拿到页面时已经是一个完整的页面了。

CSI(Client Side Include)方案：单独发起一个异步JS请求，向服务端获取内容。

5）动静分离的几种架构

1.实体机单机部署

将虚拟机改为实体机部署，以增大cache的容量，并采用一致性hash分组的方式来提升命中率。

实体机单机部署的优点：

a.没用网路瓶颈，而且能使用大内存。

b.能够提升命中率，又能减少Gzip压缩。

c.采用定时失效方式，减少cache失效压力，

2.统一cache层

统一cache层就是将单机的cache统一分离出来，形成一个单独的cache集群。将cache层拿出来统一管理可以减少运维成 本，同时方便接入其他静态化系统。

优点：

a.单独一个cache层，可以减少多个应用使用cache的成本。

b.统一cache层更易于维护。

c.可以共享内存，最大化利用内存，不同系统之间的内存可以动态切换，从而能够有效应用各种攻击。

3.上CDN

将cache移动CDN上，有几个问题需要解决：

第一，失效问题

第二，命中率问题，cache最重要的一个衡量指标就是高命中率，不然cache的存在就市区了意义。

第三，发布更新问题。

5.有针对性的处理好热点数据

1）什么是热点？

热点分为热点操作和热点数据。

热点操作：如大量刷新页面、大量添加购物车等操作，这些操作可以抽象为读操作和写操作。

热点数据：分为静态热点数据和动态热点数据。静态热点数据就是能够提前预测的数据，如通过报名的方式提前筛选出来或通过大数据分析来提前发现热点商品。动态热点数据就是系统运行过程中产生的热点。

2）发现静态热点数据

静态热点数据可以通过提前报名方式筛选出来，也可以通过技术手段提前预测，例如对买家每天访问的商品进行大数据计算，统计出TOP N的商品，可以认为这些就是热点商品。

发现静态热点数据的缺点是时效性较差。

3）发现动态热点数据

发现动态热点数据的具体实现：

1.构建一个异步的系统，可以收集交易链路上各个环节中的中间件产品的热点key，如Nginx、缓存、RPC服务框架等。

2.建立一个热点上报和可以按照需求订阅的热点服务下发规范，目的是通过交易链路上各个系统访问的时间差，把上游已 经发现的热点透传给下游系统，提前做好保护。

3. 将上游系统收集的热点数据发送到热点服务台，然后下游系统就会知道哪些商品会被频繁调用，然后做热点保护。

4）热点数据的处理

处理热点数据的思路是优化、限制、隔离

优化：优化热点数据最有效的办法就是缓存热点数据，如果热点数据做了动静分离，可以长期缓存静态数据，但热点数据更多的是临时缓存，设置缓存失效时间。

限制：限制更多的是一种保护机制，例如对商品ID做一致性hash，把热点商品限制在一个队列里，防止热点商品占用太多的服务器资源，而使其他请求得不到处理资源。

隔离：秒杀的第一原则就是将热点数据隔离出来，不要让1%的请求影响到另外99%的业务。

业务隔离：把秒杀做成一种营销活动，卖家要参加秒杀这种营销活动需要单独报名，从技术上来说可以提前做好预热。

系统隔离：系统隔离更多的是运行时的隔离，可以通过分组部署的方式和另外 99% 分开。

数据隔离：秒杀所调用的数据大部分都是热点数据，比如会启用单独的 Cache 集群或者 MySQL 数据库来放热点数据。

6.流量削峰

削峰的存在可以让服务器的处理变得更加平稳，可以节省服务器的资源成本。削峰本质上就是更多的延缓用户请求的发出，以便减少和锅炉一些无效请求。

流量削峰的几种操作思路：排队、答题、分层过滤。

1）排队

用消息队列来缓冲瞬时流量，除了消息队列，类似排队还有很多方式，如：

a.利用线程池加锁等待也是一种常用排队方式。

b.先进先出、先进后出等常用的内存排队算法的实现方式。

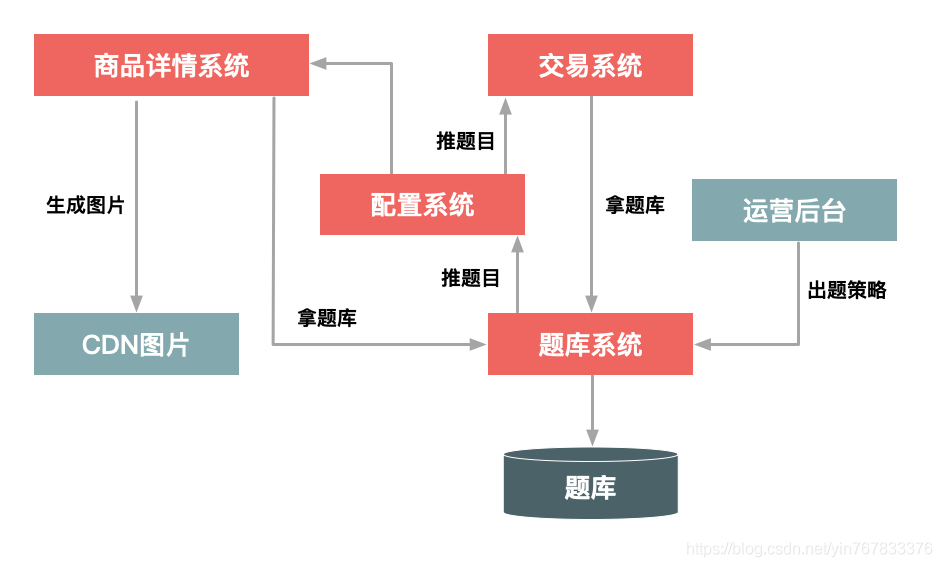
c.把请求序列化到文件中，然后再顺序的读文件来恢复请求等方式。

这些方式都是把一步的操作变成两步的操作，起到缓冲的作用。

2）答题

增加答题有两个目的，一个是防止部分买家使用秒杀器在参加秒杀时作弊。另一个目的是延缓请求，起到对请求流量进行削峰的作用，从而让系统更好的支持瞬时的流量高峰。

秒杀答题设计思路：



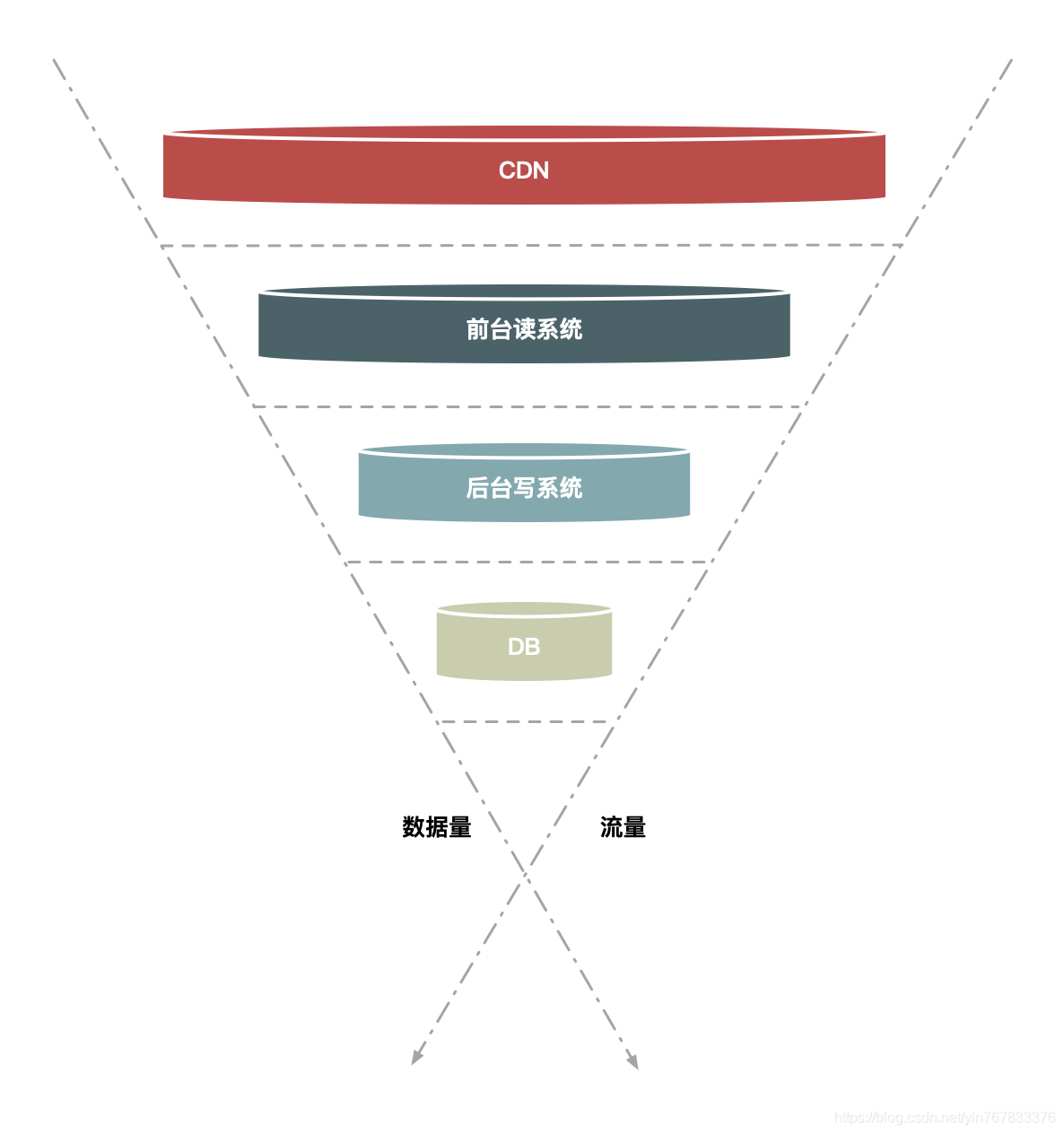
整个答题逻辑主要分为3部分

第一，题库生成模块，主要是生成一个问题和答案，防止秒杀器来答题。

第二，题库的推送模块，用于在答题前，把题目推送给详情系统和交易系统。题库推送主要为了保证每次用户请求的题目是唯一的，防止答题作弊。

第三，题目的图片生成模板，用于把题目生成图片格式，并在图片里增加干扰因素。由于答题时网络拥挤，应该把题目图片提前推送到CDN上并进行预热。

3）分层过滤



核心思想是：在不同的层次尽可能的过滤掉无效请求，让漏斗最末端的才是有效请求，要达到这种效果，必须对数据做分层校验。

分层校验原则：

a.将动态请求的读数据缓存在web端，过滤掉无效的数据读。

b.对读数据不做强一致性校验，减少因为一致性校验产生瓶颈的问题。

c.对写数据进行基于时间的合理分片，过滤掉过期的失效请求。

d.对写请求做限流保护，将超出系统承载能力的请求过滤掉。

e.对写数据进行强一致性校验，只保留最后有效的数据。

分层校验的目的是：在读系统中，尽量减少由于一致性校验带来的系统瓶颈，尽量将不影响性能的校验提前。在写数据系统中，主要对写的数据做一致性检查，最后在数据库层保证数据的最终准确性。

7.影响性能的因素

主要讨论影响系统服务端性能的因素，一般用QPS（Query Per Second，每秒请求数）衡量，与QPS相关的一个影响就是响应时间（Response Time，RT），可以理解为服务器处理响应的耗时。

8.如何优化系统

1）减少编码

java编码运行比较慢，在很多场景下，涉及字符串的操作都比较耗CPU资源，不管是磁盘I/O还是网络I/O，都需要将字符转换成字节，而转换必须编码，每个字符的编码都需要查表，这种查表操作非常耗资源，所以减少字符到字节或相反的转换、减少字符编码会非常有效，减少编码可以大大提升性能。

2）减少序列化

减少java中的序列化操作可以大大提升系统性能。序列化大部分是在 RPC 中发生的，因此避免或者减少 RPC 就可以减少序列化。有一种新的方案，就是可以将多个关联性比较强的应用进行“合并部署”，而减少不同应用之间的 RPC 也可以减少序列化的消耗。

3）java极致优化

ava 和通用的 Web 服务器（如 Nginx 或 Apache 服务器）相比，在处理大并发的 HTTP 请求时要弱一点，所以一般我们都会对大流量的 Web 系统做静态化改造，让大部分请求和数据直接在 Nginx 服务器或者 Web 代理服务器上直接返回，而 Java 层只需处理少量数据的动态请求。针对这些请求的优化：

a.直接使用 Servlet 处理请求,避免使用传统的 MVC 框架，这样可以绕过一大堆复杂且用处不大的处理逻辑.

b.直接输出流数据，使用 resp.getOutputStream() 而不是 resp.getWriter() 函数，可以省掉一些不变字符数据的编码，从而提升性能；数据输出时推荐使用 JSON 而不是模板引擎来输出页面。

4）并发读优化

采用应用层的 LocalCache，即在秒杀系统的单机上缓存商品相关的数据。需要划分成动态数据和静态数据分别进行处理：

a.像商品中的“标题”和“描述”这些本身不变的数据，会在秒杀开始之前全量推送到秒杀机器上，并一直缓存到秒杀结束；

b.库存这类动态数据，会采用“被动失效”的方式缓存一定时间，失效后再去缓存拉取最新的数据。

9.减库存的几种方式

1）下单减库存：下单减库存是最简单的减库存方式，也是控制最精确的一种，下单时直接通过数据库的事务机制控制商品库 存，这样一定不会出现超卖的情况。

2）付款减库存：付款时才减库存，如果并发比较高，有可能出现买家下单后付不了款的情况，因为商品可能已经卖出。

3）预扣库存：买家下单后，库存为其保留一定的时间，超过这个时间，库存将会自动释放，释放后其他买家就可以继续购买。

如何减库存？

第一种，在应用中通过事务来判断，保证减后库存不能为负数，否则就回滚。

第二种，直接设置数据库字段数据为无符号整数，这样减后库存字段值小于零时执行SQL语句会报错

第三种，通过SQL的case when语句判断或乐观锁实现。

10.减库存的极致优化

减库存需要再数据库层中完成操作，由于MySQL 存储数据的特点，同一数据在数据库里肯定是一行存储（MySQL），因此会有大量线程来竞争 InnoDB 行锁，而并发度越高时等待线程会越多，TPS（Transaction Per Second，即每秒处理的消息数）会下降，响应时间（RT）会上升，数据库的吞吐量就会严重受影响。

分离热点商品到单独的数据库没有解决并发锁的问题，要解决并发锁有两种办法：

1）应用层做排队。按照商品维度设置队列顺序执行，这样能减少同一台机器对数据库同一行记录进行操作的并发度，同时也能控制单个商品对数据库连接的数据，防止热点商品占用太多的数据库连接。

2）数据库层做排队。应用层只能做到单机排队，控制并发的能力有限，如果能在数据库层做全局排队是最理想的。阿里针对这种MySQL的innoDB层上的补丁程序（patch），可以再数据库层上对单行记录做到并发排队。

11.遇到大流量时，如何保障系统的稳定运行

1）降级

当系统的容量达到一定程度时，限制或者关闭系统的某些非核心功能，从而把有限的资源保留给更核心的业务。它是一个有目的、有计划的执行过程，所以对降级我们一般需要有一套预案来配合执行。如果我们把它系统化，就可以通过预案系统和开关系统来实现降级。

执行降级无疑是在系统性能和用户体验之间选择了前者，降级后肯定会影响一部分用户的体验。所以降级的核心目标是牺牲次要的功能和用户体验来保证核心业务流程的稳定，是一个不得已而为之的举措。

2）限流

限流就是当系统容量达到瓶颈时，我们需要通过限制一部分流量来保护系统，并做到既可以人工执行开关，也支持自动化保护的措施。限流可以再客户端，也可以再服务端。

客户端限流，好处是可以限制请求的发出，通过减少发出无用请求从而减少对系统的消耗。缺点是当客户端比较分散时，没法设置合理的限流阈值，如果阈值设置的太小，会导致服务端没有达到瓶颈时客户端已经被限制，如果设置的太大，起不到限制的作用。

服务端限流，好处是可以根据服务端的性能设置合理的阈值，缺点是被限制的请求都是无效的请求，处理这些无效请求本身也会消耗服务器资源。

限流会影响用户的正常请求，必然会导致一部分用户请求失败，因此在系统处理这种异常时一定要设置超时时间，防止因被限流的请求不能fast fail(快速失败)而拖垮系统。

3）拒绝服务

当系统负载达到一定阈值时，例如 CPU 使用率达到 90% 或者系统 load 值达到 2\*CPU 核数时，系统直接拒绝所有请求，这种方式是最暴力但也最有效的系统保护方式。拒绝服务用以防止最坏情况发生，防止因把服务器压垮而长时间彻底无法提供服务，像这种系统过载保护虽然在过载时无法提供服务，但是系统仍然可以运作，当负载下降时又很容易恢复，所以每个系统和每个环节都应该设置这个兜底方案，对系统做最坏情况下的保护。

组件分支：

组件分支

秒杀系统组件

对一些高并发的操作进行管道操作

【作为特殊组件可以抽离系统】

《外部系统》

大型机银行系统

金额交易

全局变量控制器

封装账户的全局变量

账户总控制器

判断账户的全局总览信息

购物系统组件

对用户进行购物商品的交易

转账系统组件

对用户进行金额转账

非关系数据库[Redis]

存储静态全局便变量

中间缓存件

为用户交易，页面动态信息，全局变量提供快速封装

数据库【MySql】

存储用户信息，访问日志

安全组件

为用户组测 ，全局信息，改密码提供安全能力

Web应用

通过浏览器范文互联网系统

转账系统组件

对用户进行金额转账

非关系数据库[Redis]

对商品的名称和商品链接进行map缓存，加快查找速度

秒杀系统组件

对一些高并发的操作进行管道操作

【作为特殊组件可以抽离系统】

下单组件

【orderComponent】

在给商品上锁后在用户下单中添加一个可购买判断

商品组件

【produceComponent】

在每个购买商品时给商品上锁在进行用户的订单添加商品

系统控制器[shopController]

通过后端方法对用户购物系统控制

购物系统

释放锁

加锁

回滚

《外部系统》

大型机银行系统

金额交易

转账交易组件

【tranlateComponent】

用户金额转账

秒杀系统组件

对一些高并发的操作进行管道操作

【作为特殊组件可以抽离系统】

转账系统

系统控制器[tranlateController]

通过后端方法对用户转账系统控制

用户抢购指定[shopProtoUser]

给用户一个购买商品的锁

队列提取组件[DrawQueue]

对队列内的进行提取，将其中选中用户进行进一步操作

缓存中间件

购物组件

转账组件

红包账户平帐

[RedBlankCommit]

将奖金红包转入用户之中

秒杀系统

队列组件[Intoqueue]

对访问的人员单独序列号进行队列填充

秒杀商品控制器

[shopProduceCrontroll]

控制后端队列

CDN

静态Cache数据

秒杀页面

动态数据保存

秒杀红包控制器

[redbegController]

控制后端队列

JSON/HTTPS

**抢红包页面端**

**通过浏览器访问秒杀系统的控制台入口**