#### **网站优化**

#### **Tomcat 优化**

Tomcat支持以下三种模式：

---BIO：一个线程处理一个请求，缺点：并发量高时，线程数较多，浪费资源，

Tomcat7或以下在Linux系统中默认使用这种方式。

---NIO：利用Java的异步IO处理，可以通过少量的线程处理大量的请求。

Tomcat8在Linux系统中默认使用这种方式。Tomcat7 必须修改Connector配

置来启用。

---APR(Apache Portable Runtime)：从操作系统层面解决io阻塞问题。Linux

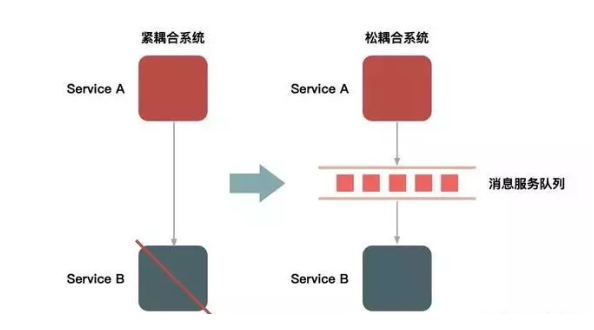
如果安装了apr和native，Tomcat直接启动就支持apr。

#### **削峰**

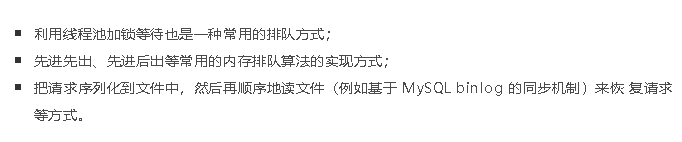
削峰的存在，一是可以让服务端处理变得更加平稳，二是可以节省服务器的资源成本。针对秒杀这一场景，削峰从本质上来说就是更多地延缓用户请求的发出，以便减少和过滤掉一些无效请求，它遵从“请求数要尽量少”的原则。

**排队**

用消息队列来缓冲瞬时流量，把同步的直接调用 转换成异步的间接推送，中间通过一个队列在一端承接瞬时的流量洪峰，在另一端平滑地将消息 推送出去。



其他方式：



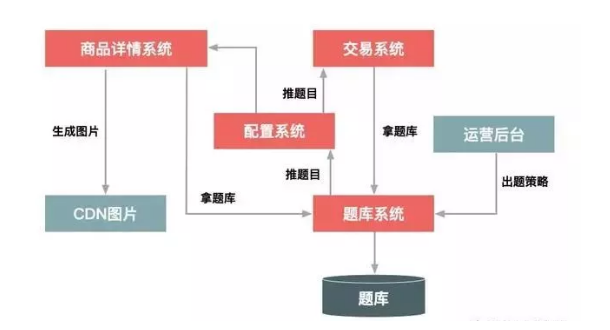
可以看到，这些方式都有一个共同特征，就是把“一步的操作”变成“两步的操作”，其中增加 的一步操作用来起到缓冲的作用。

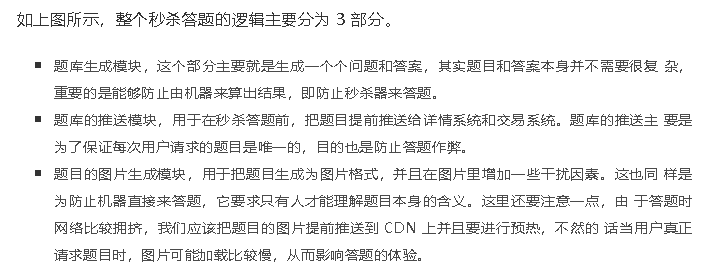
**答题**

第一个目的是防止部分买家使用秒杀器在参加秒杀时作弊。

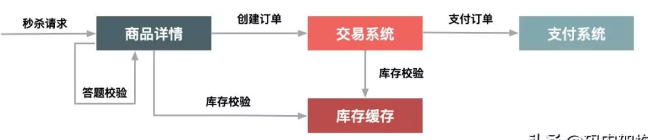
第二个目的其实就是延缓请求，起到对请求流量进行削峰的作用，从而让系统能够更好地支持瞬 时的流量高峰。这个重要的功能就是把峰值的下单请求拉长，从以前的 1s 之内延长到 2s~10s。这样一来，请求峰值基于时间分片了。这个时间的分片对服务端处理并发非常重要，会大大减轻 压力。而且，由于请求具有先后顺序，靠后的请求到来时自然也就没有库存了，因此根本到不了 最后的下单步骤，所以真正的并发写就非常有限了。

**案例：**



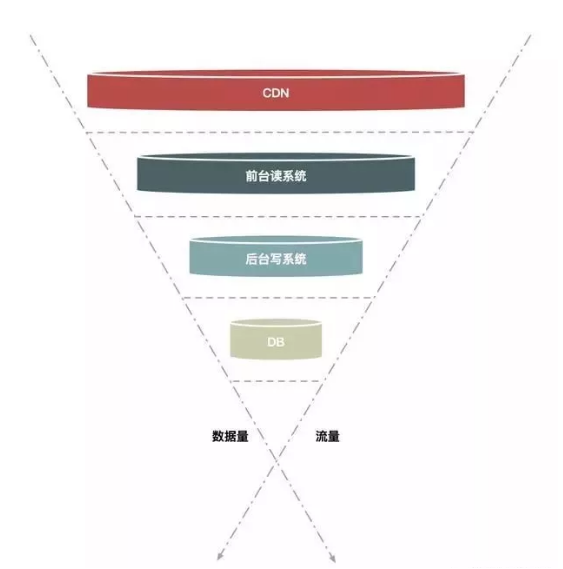


验证的逻辑如下图：



这里面的验证逻辑，除了验证问题的答案以外，还包括用户本身身份的验证，例如是否已 经登录、用户的 Cookie 是否完整、用户是否重复频繁提交等。

**分层过滤**



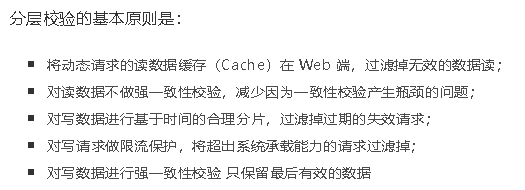
1、假如请求分别经过 CDN、前台读系统（如商品详情系统）、后台系统（如交易系统）和数据库 这几层，那么：

2、大部分数据和流量在用户浏览器或者 CDN 上获取，这一层可以拦截大部分数据的读取；

3、经过第二层（即前台系统）时数据（包括强一致性的数据）尽量得走 Cache，过滤一些无效的 请求；

4、再到第三层后台系统，主要做数据的二次检验，对系统做好保护和限流，这样数据量和请求就 进一步减少；

5、最后在数据层完成数据的强一致性校验。



**总结**

队列缓冲方式更加通用，它适用于内部上下游系统之间调用请求不平缓的场景，由于内部 系统的服务质量要求不能随意丢弃请求，所以使用消息队列能起到很好的削峰和缓冲作用。

而答题更适用于秒杀或者营销活动等应用场景，在请求发起端就控制发起请求的速度，因为越到 后面无效请求也会越多，所以配合后面介绍的分层拦截的方式，可以更进一步减少无效请求对系 统资源的消耗。

分层过滤非常适合交易性的写请求，比如减库存或者拼车这种场景，在读的时候需要知道还有没 有库存或者是否还有剩余空座位。但是由于库存和座位又是不停变化的，所以读的数据是否一定 要非常准确呢？其实不一定，你可以放一些请求过去，然后在真正减的时候再做强一致性保证， 这样既过滤一些请求又解决了强一致性读的瓶颈。

在削峰的处理方式上除了采用技术手段，其实还可以采用业务手段来达到一定效果，例如 在零点开启大促的时候由于流量太大导致支付系统阻塞，这个时候可以采用发放优惠券、发起抽 奖活动等方式，将一部分流量分散到其他地方，这样也能起到缓冲流量的作用。