高并发秒杀接口优化

秒杀业务场景，并发量很大，瓶颈在数据库，怎么解决，**加缓存**。用户发起请求时，从浏览器开始,在浏览器上做**页面静态化**直接将页面缓存到用户的浏览器端，然后请求到达网站之前可以部署CDN节点，让请求先访问CDN，到达网站时候使用页面缓存。页面缓存再进一步的话，粒度再细一点的话就是**对象缓存**。缓存层依次请求完之后，才是数据库。通过一层一层的访问缓存逐步的**削减到达数据库的请求数量**，这样才能保证网站在大并发之下抗住压力。

但是仅仅依靠缓存还不够，所以还需要进行**接口优化**。
接口优化核心思路：减少数据库的访问。（数据库抗并发的能力有限）

- 使用**Redis预减库存**减少对数据库的访问
- 使用**内存标记减少Redis的访问**
- 使用**RabbitMQ队列缓冲**，异步下单，增强用户体验

具体实现步骤：

1. 系统初始化，把商品**库存数量加载到Redis**上面来

   *MiaoshaController*实现*InitializingBean*接口，重写*afterPropertiesSet*方法。在容器启动的时候，检测到了实现了接口*InitializingBean*之后，就回去回调*afterPropertiesSet*方法。将每种商品的库存数量加载到redis里面去。

2. 收到请求，Redis**预减库存（先减少Redis里面的库存数量，库存不足，直接返回）**，如果库存已经到达临界值的时候，即=0，就不需要继续往下发送请求，直接返回失败, 如果库存充足，且无重复秒杀，将秒杀请求封装后消息入队，

3. 请求入队，立即返回**排队中**

   注意：消息队列这里，**消息只能传字符串**，*MiaoshaMessage* 这里是个Bean对象，是先用*beanToString*方法，将转换为*String*，放入队列，使用AmqpTemplate传输。

   同时给前端返回一个code (0)，即代表返回排队中。（返回的并不是失败或者成功，此时还不能判断）

4. 请求出队，**生成订单，减少库存**

   后端*RabbitMQ*监听秒杀*MIAOSHA_QUEUE*的名字的通道，如果有消息过来，获取到传入的信息，执行真正的秒杀之前，要判断数据库的库存，判断是否重复秒杀，然后执行秒杀事务（秒杀事务是一个原子操作(数据库事务)：库存减1，下订单，写入秒杀订单）。**注意：秒杀操作是一个事务，使用@Transactional注解来标识，如果减少库存失败，则回滚。**

5. 客户端**轮询**，是否秒杀成功

   此时，前端根据商品id轮询请求接口*MiaoshaResult*,查看是否生成了商品订单，如果请求返回-1代表秒杀失败，返回0代表排队中，返回>0代表商品id说明秒杀成功。