## 登录注册：

关于登录，我们当时考虑到以后用户量增大情况，我们队数据库中的表进行了分表。从来来提高查询的效率，这个分表的方法可以是时间，id等；

用户注册，首先需要用户手机号的验证，这时用户填写手机号后，我们会先将用户手机号和验证码保存在缓存中，并且是要设置获过期时间，一般为15分钟，然后调用短信接口将随机生成的验证码发送给用户，当用户输入验证码点击提交时，我们会根据用户手机号去缓存中进行查询，如果验证码正确，才能跳转到填写账号信息页面 ，进行账号的填写；这是用户填写用户名时，在失去焦点是我们会进行一个验证。查看数据库中是否存在这个用户名，如果存在就提示该账户以存在，请更换用户名，还有两次的密码都必须一致可以，

登录时用户可以根据手机号，用户名，邮箱进行登录，首先因为这些都是具有唯一性的，不会存在重复的情况，所以我们都可用来登录；只要有一个条件错误，我们就会告诉用户，用户名和密码不匹配，并且提醒用户是忘记密码或者忘记用户名，然后可以进行找回；

除了正常的登录注册外，当时我们还想到了单账号登录这个功能；具体原理是：当用户登录之后，我们会以当前用户信息(唯一标示id，code等)为键，而sessionid为值，来存放在缓存中，只要浏览器不关闭，或者不跨浏览器和电脑，sessionid是相同的；

除此之外，我们还会以sessionid为键，用户信息（全部信息）为值也存放在redis中；这个是为了拦截器的使用，springmvc的拦截器是在每次进入控制层之前执行一次方法，在这个方法中，我们通过session去redis中查，如果存在对象，说明是在登录状态，我们就会让其执行方法(return true )，如果不存在，还要走一些我们需要拦截的方法，我们就会让其返回到登录页面(return false)；

如果现在在别的地方也登录同一账户，我们首先会先获取当前sessionid ，第二我们会通过用户信息(唯一标示id，code等)去redis中获取以前存放的sessionid，如果两次sessionid相同说明在同一浏览器，如果不相同，说明在不同的浏览器；我们就会删除掉原来在redis中存放的数据，将新的数据存放进去，这样原来浏览器在执行其他操作的时候就会被拦截到登录页面。

## 每秒处理十万订单的支付系统思路

支付系统要处理每秒十万笔订单，需要的是每秒数十万的数据库更新操作（insert加update），这在任何一个独立数据库上都是不可能完成的任务，所以我们首先要做的是对订单表（简称order）进行分库与分表。

分库策略我们选择了“二叉树分库”，所谓“二叉树分库”指的是：我们在进行数据库扩容时，都是以2的倍数进行扩容。并将每个分库中都将order表拆分成10份。

如果要支持每秒处理10万订单，那每秒将至少需要生成10万个订单ID，通过数据库生成自增ID显然无法完成上述要求。所以我们只能通过内存计算获得全局唯一的订单ID。使用Twitter的Snowflake算法，实现了全局唯一ID。订单id由分库分表信息+时间戳+机器号+自增序号组成。这样只根据订单ID，我们也能快速的查询到对应的订单信息。

我们通过对order表uid维度的分库分表，实现了order表的超高并发写入与更新，并能通过uid和订单ID查询订单信息。但作为一个开放的集团支付系统，我们还需要通过业务线ID（又称商户ID，简称bid）来查询订单信息，所以我们引入了bid维度的order表集群，将uid维度的order表集群冗余一份到bid维度的order表集群中，要根据bid查询订单信息时，只需查bid维度的order表集群即可。

我们引入了消息队列进行异步数据同步，来实现数据的最终一致性。当然消息队列的各种异常也会造成数据不一致，所以我们又引入了实时监控服务，实时计算两个集群的数据差异，并进行一致性同步。

没有任何机器或服务能保证在线上稳定运行不出故障。比如某一时间，某一数据库主库宕机，这时我们将不能对该库进行读写操作，线上服务将受到影响。

所谓数据库高可用指的是：当数据库由于各种原因出现问题时，能实时或快速的恢复数据库服务并修补数据，从整个集群的角度看，就像没有出任何问题一样。需要注意的是，这里的恢复数据库服务并不一定是指修复原有数据库，也包括将服务切换到另外备用的数据库。数据库高可用的主要工作是数据库恢复与数据修补

支付系统除了最核心的支付订单表与支付流水表外，还有一些配置信息表和一些用户相关信息表。如果所有的读操作都在数据库上完成，系统性能将大打折扣，所以我们引入了数据分级机制。

我们简单的将支付系统的数据划分成了3级：

1级：订单数据和支付流水数据；这两块数据对实时性和精确性要求很高，所以不添加任何缓存，读写操作将直接操作数据库。

第2级：用户相关数据；这些数据和用户相关，具有读多写少的特征，所以我们使用redis进行缓存。

第3级：支付配置信息；这些数据和用户无关，具有数据量小，频繁读，几乎不修改的特征，所以我们使用本地内存进行缓存。

http请求在进入web集群前，会先经过一层粗细管道。入口端是粗口，入口端是粗口，我们设置最大能支持100万请求每秒，多余的请求会被直接抛弃掉。出口端是细口，我们设置给web集群10万请求每秒。剩余的90万请求会在粗细管道中排队，等待web集群处理完老的请求后，才会有新的请求从管道中出来，给web集群处理。这样web集群处理的请求数每秒永远不会超过10万，在这个负载下，集群中的各个服务都会高校运转，整个集群也不会因为暴增的请求而停止服务。