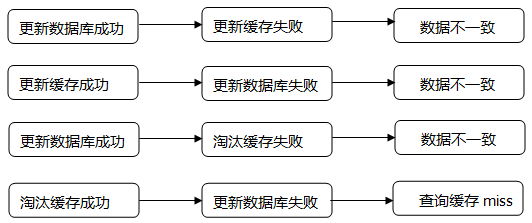
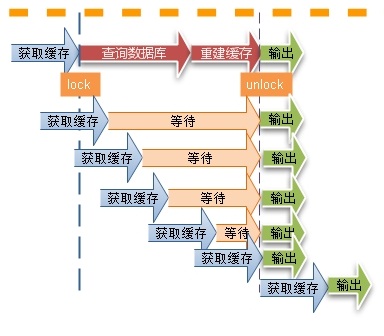
**缓存一致性问题**

当数据时效性要求很高时，需要保证缓存中的数据与数据库中的保持一致，而且需要保证缓存节点和副本中的数据也保持一致，不能出现差异现象。这就比较依赖缓存的过期和更新策略。一般会在数据发生更改的时，主动更新缓存中的数据或者移除对应的缓存。



**缓存并发问题**

缓存过期后将尝试从后端数据库获取数据，这是一个看似合理的流程。但是，在高并发场景下，有可能多个请求并发的去从数据库获取数据，对后端数据库造成极大的冲击，甚至导致 “雪崩”现象。此外，当某个缓存key在被更新时，同时也可能被大量请求在获取，这也会导致一致性的问题。那如何避免类似问题呢？我们会想到类似“锁”的机制，在缓存更新或者过期的情况下，先尝试获取到锁，当更新或者从数据库获取完成后再释放锁，其他的请求只需要牺牲一定的等待时间，即可直接从缓存中继续获取数据。



**缓存穿透问题**

缓存穿透在有些地方也称为“击穿”。很多朋友对缓存穿透的理解是：由于缓存故障或者缓存过期导致大量请求穿透到后端数据库服务器，从而对数据库造成巨大冲击。

这其实是一种误解。真正的缓存穿透应该是这样的：

在高并发场景下，如果某一个key被高并发访问，没有被命中，出于对容错性考虑，会尝试去从后端数据库中获取，从而导致了大量请求达到数据库，而当该key对应的数据本身就是空的情况下，这就导致数据库中并发的去执行了很多不必要的查询操作，从而导致巨大冲击和压力。

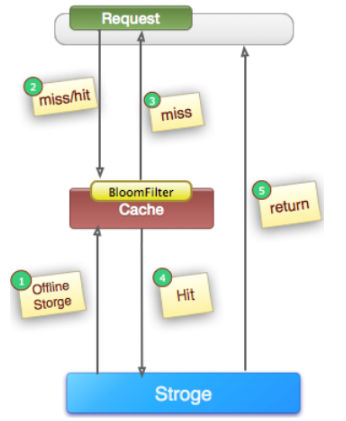
可以通过下面的几种常用方式来避免缓存传统问题：

缓存空对象

对查询结果为空的对象也进行缓存，如果是集合，可以缓存一个空的集合（非null），如果是缓存单个对象，可以通过字段标识来区分。这样避免请求穿透到后端数据库。同时，也需要保证缓存数据的时效性。这种方式实现起来成本较低，比较适合命中不高，但可能被频繁更新的数据。

单独过滤处理

对所有可能对应数据为空的key进行统一的存放，并在请求前做拦截，这样避免请求穿透到后端数据库。这种方式实现起来相对复杂，比较适合命中不高，但是更新不频繁的数据。



**缓存颠簸问题**

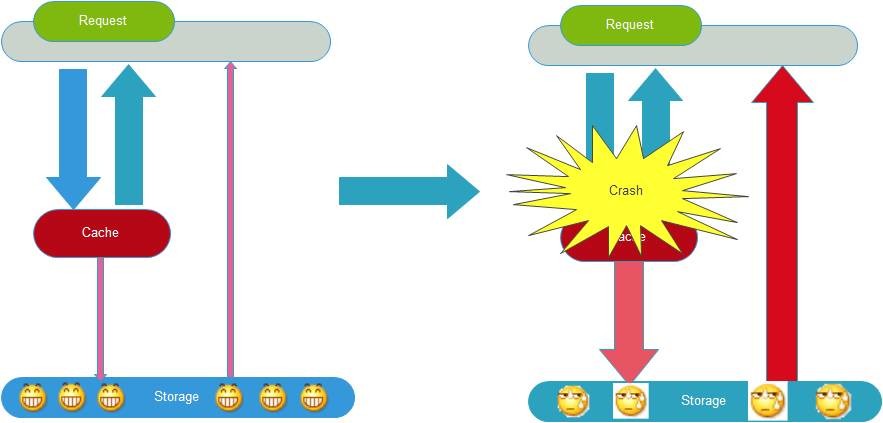
缓存的颠簸问题，有些地方可能被成为“缓存抖动”，可以看做是一种比“雪崩”更轻微的故障，但是也会在一段时间内对系统造成冲击和性能影响。一般是由于缓存节点故障导致。业内推荐的做法是通过一致性Hash算法来解决。这里不做过多阐述，可以参照其他章节

**缓存的雪崩现象**

缓存雪崩就是指由于缓存的原因，导致大量请求到达后端数据库，从而导致数据库崩溃，整个系统崩溃，发生灾难。导致这种现象的原因有很多种，上面提到的“缓存并发”，“缓存穿透”，“缓存颠簸”等问题，其实都可能会导致缓存雪崩现象发生。这些问题也可能会被恶意攻击者所利用。还有一种情况，例如某个时间点内，系统预加载的缓存周期性集中失效了，也可能会导致雪崩。为了避免这种周期性失效，可以通过设置不同的过期时间，来错开缓存过期，从而避免缓存集中失效。

从应用架构角度，我们可以通过限流、降级、熔断等手段来降低影响，也可以通过多级缓存来避免这种灾难。

此外，从整个研发体系流程的角度，应该加强压力测试，尽量模拟真实场景，尽早的暴露问题从而防范。

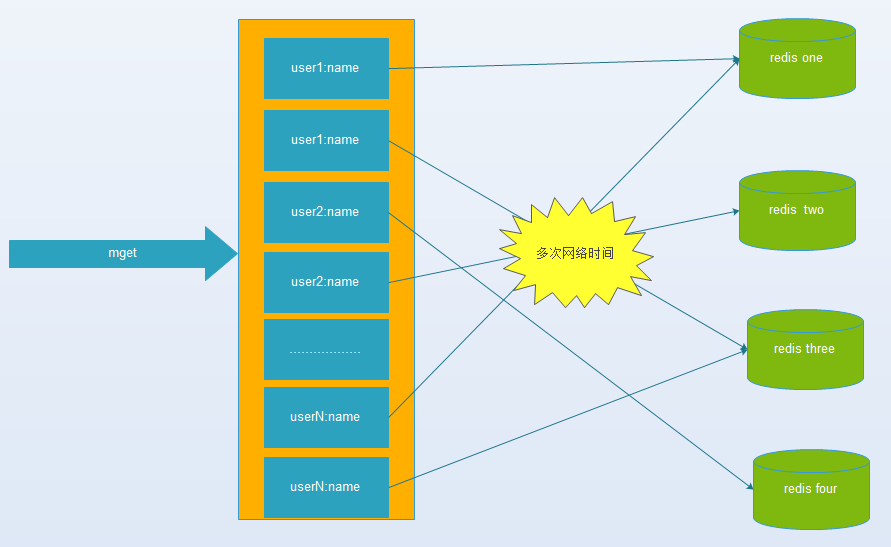


**缓存无底洞现象**

该问题由 facebook 的工作人员提出的， facebook 在 2010 年左右，memcached 节点就已经达3000 个，缓存数千 G 内容。

他们发现了一个问题---memcached 连接频率，效率下降了，于是加 memcached 节点，

添加了后，发现因为连接频率导致的问题，仍然存在，并没有好转，称之为”无底洞现象”。



目前主流的数据库、缓存、Nosql、搜索中间件等技术栈中，都支持“分片”技术，来满足“高性能、高并发、高可用、可扩展”等要求。有些是在client端通过Hash取模（或一致性Hash）将值映射到不同的实例上，有些是在client端通过范围取值的方式映射的。当然，也有些是在服务端进行的。但是，每一次操作都可能需要和不同节点进行网络通信来完成，实例节点越多，则开销会越大，对性能影响就越大。

主要可以从如下几个方面避免和优化：

数据分布方式

有些业务数据可能适合Hash分布，而有些业务适合采用范围分布，这样能够从一定程度避免网络IO的开销。

IO优化

可以充分利用连接池，NIO等技术来尽可能降低连接开销，增强并发连接能力。

数据访问方式

一次性获取大的数据集，会比分多次去获取小数据集的网络IO开销更小。

当然，缓存无底洞现象并不常见。在绝大多数的公司里可能根本不会遇到。