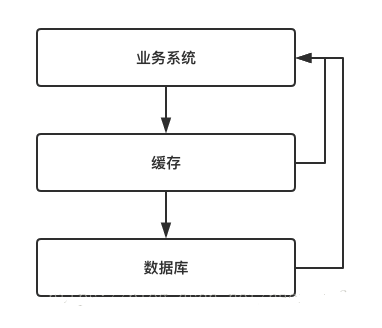
## 缓存的应用场景

在大多数互联网应用中，缓存的使用方式如下图所示：



1. 当业务系统发起某一个查询请求时，首先判断缓存中是否有该数据；
2. 如果缓存中存在，则直接返回数据；
3. 如果缓存中不存在，则再查询数据库，然后返回数据。

## 缓存穿透

### 1.2.1 什么是缓存穿透？

业务系统要查询的数据根本就存在！当业务系统发起查询时，按照上述流程，首先会前往缓存中查询，由于缓存中不存在，然后再前往数据库中查询。由于该数据压根就不存在，因此数据库也返回空。这就是缓存穿透。

综上所述：业务系统访问压根就不存在的数据，就称为缓存穿透

### 1.2.2 为什么会发生缓存穿透

发生缓存穿透的原因有很多，一般为如下两种：

恶意攻击，故意营造大量不存在的数据请求我们的服务，由于缓存中并不存在这些数据，因此海量请求均落在数据库中，从而可能会导致数据库崩溃。

代码逻辑错误。这是程序员的锅，没啥好讲的，开发中一定要避免！

### 1.2.3缓存穿透的解决方案

1缓存空数据

之所以发生缓存穿透，是因为缓存中没有存储这些空数据的key，导致这些请求全都打到数据库上。

那么，我们可以稍微修改一下业务系统的代码，将数据库查询结果为空的key也存储在缓存中。当后续又出现该key的查询请求时，缓存直接返回null，而无需查询数据库。

缓存空对象会有两个问题：

第一，空值做了缓存，意味着缓存层中存了更多的键，需要更多的内存空间 ( 如果是攻击，问题更严重 )，比较有效的方法是针对这类数据设置一个较短的过期时间，让其自动剔除。

第二，缓存层和存储层的数据会有一段时间窗口的不一致，可能会对业务有一定影响。例如过期时间设置为 5 分钟，如果此时存储层添加了这个数据，那此段时间就会出现缓存层和存储层数据的不一致，此时可以利用消息系统或者其他方式清除掉缓存层中的空对象。

1.4.2 BloomFilter

第二种避免缓存穿透的方式即为使用BloomFilter（布隆过滤器）。

1.4.3 两种方案的比较

这两种方案都能解决缓存穿透的问题，但使用场景却各不相同。

对于一些恶意攻击，查询的key往往各不相同，而且数据贼多。此时，第一种方案就显得提襟见肘了。因为它需要存储所有空数据的key，而这些恶意攻击的key往往各不相同，而且同一个key往往只请求一次。因此即使缓存了这些空数据的key，由于不再使用第二次，因此也起不了保护数据库的作用。

因此，对于空数据的key各不相同、key重复请求概率低的场景而言，应该选择第二种方案。而对于空数据的key数量有限、key重复请求概率较高的场景而言，应该选择第一种方案。

## 缓存雪崩

如果缓存因某种原因发生了宕机，那么原本被缓存抵挡的海量查询请求就会像疯狗一样涌向数据库。此时数据库如果抵挡不了这巨大的压力，它就会崩溃。这就是缓存雪崩。

### 如何避免缓存雪崩？

2.2.1 使用缓存集群，保证缓存高可用

和飞机都有多个引擎一样，如果缓存层设计成高可用的，即使个别节点、个别机器、甚至是机房宕掉，依然可以提供服务，例如前面介绍过的 Redis Sentinel 和 Redis Cluster 都实现了高可用。

### 使用Hystrix

Hystrix是一款开源的“防雪崩工具”，它通过 熔断、降级、限流三个手段来降低雪崩发生后的损失。

Hystrix就是一个Java类库，它采用命令模式，每一项服务处理请求都有各自的处理器。所有的请求都要经过各自的处理器。处理器会记录当前服务的请求失败率。一旦发现当前服务的请求失败率达到预设的值，Hystrix将会拒绝随后该服务的所有请求，直接返回一个预设的结果。这就是所谓的“熔断”。当经过一段时间后，Hystrix会放行该服务的一部分请求，再次统计它的请求失败率。如果此时请求失败率符合预设值，则完全打开限流开关；如果请求失败率仍然很高，那么继续拒绝该服务的所有请求。这就是所谓的“限流”。而Hystrix向那些被拒绝的请求直接返回一个预设结果，被称为“降级”。

## 缓存击穿

### 什么是热点数据集中失效？

我们一般都会给缓存设定一个失效时间，过了失效时间后，该数据库会被缓存直接删除，从而一定程度上保证数据的实时性。

但是，对于一些请求量极高的热点数据而言，一旦过了有效时间，此刻将会有大量请求落在数据库上，从而可能会导致数据库崩溃

如果某一个热点数据失效，那么当再次有该数据的查询请求[req-1]时就会前往数据库查询。但是，从请求发往数据库，到该数据更新到缓存中的这段时间中，由于缓存中仍然没有该数据，因此这段时间内到达的查询请求都会落到数据库上，这将会对数据库造成巨大的压力。此外，当这些请求查询完成后，都会重复更新缓存。

### 解决方案

1. 双重验证+同步代码块