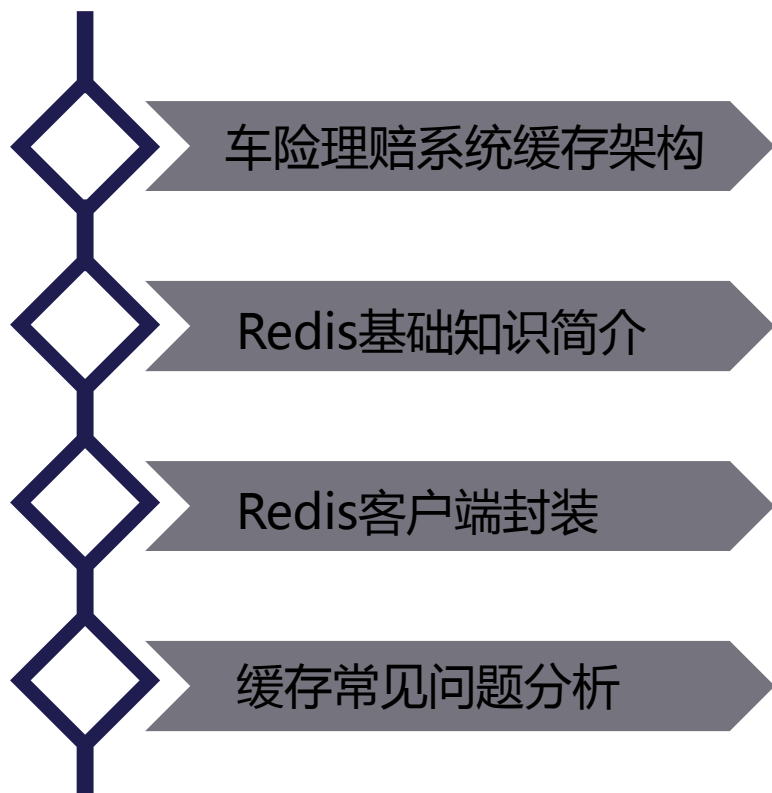


Redis缓存方案分享

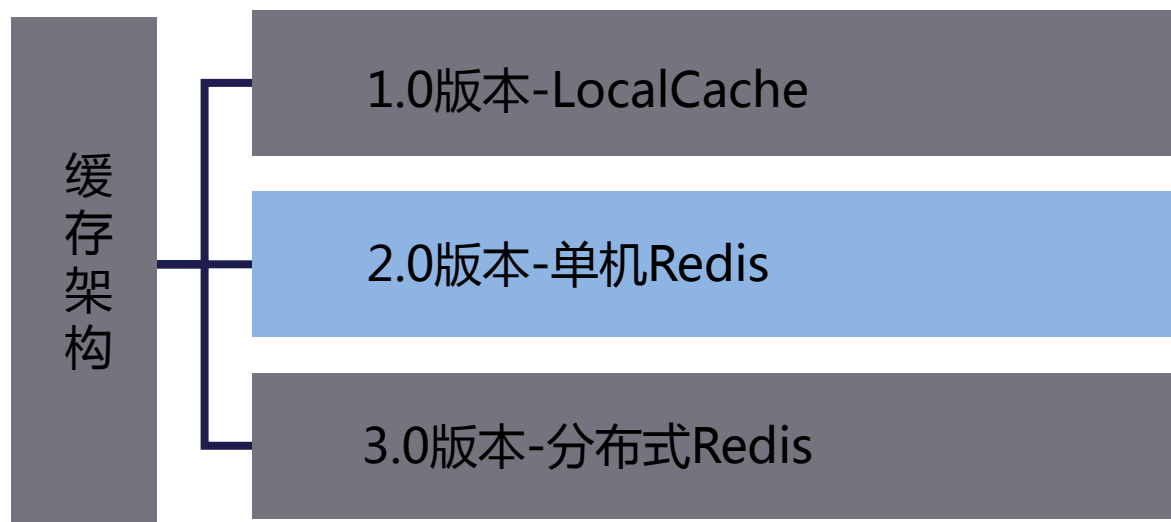
服务团队技术分享
--冯宝兴



目录



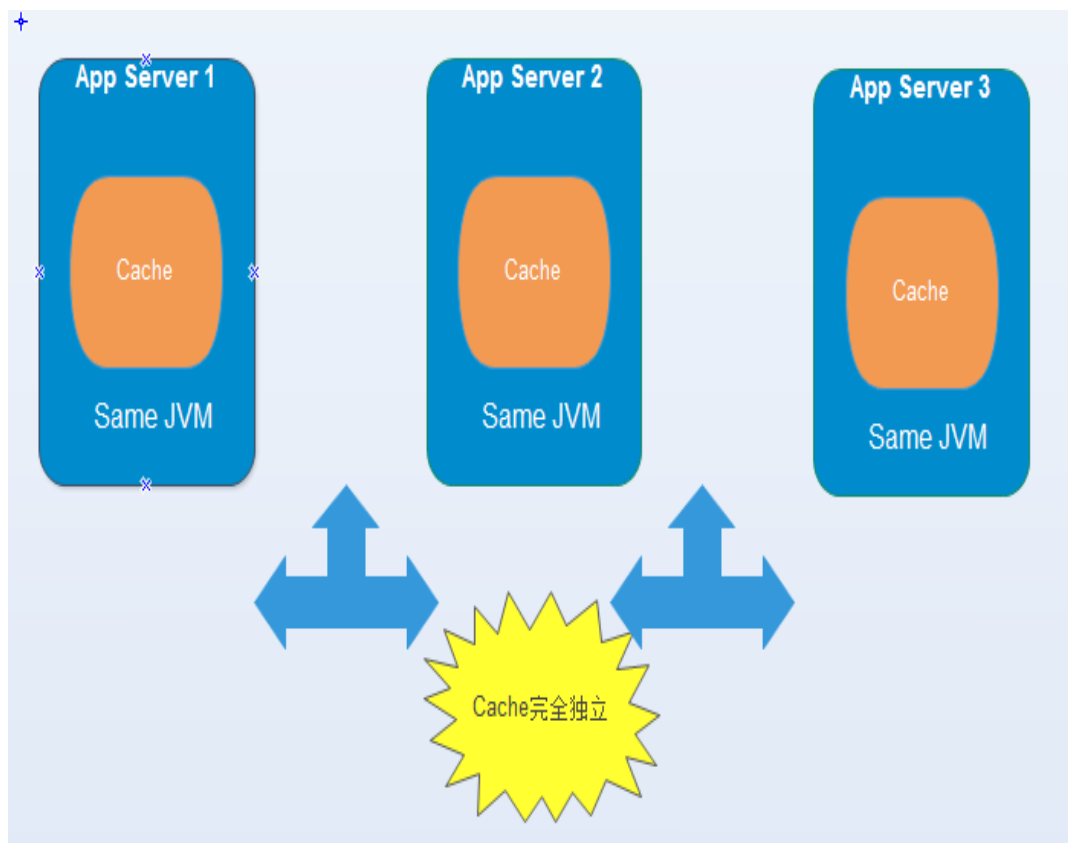
一、车险理赔系统缓存架构



车险理赔系统缓存1.0

LocalCache(独立式):

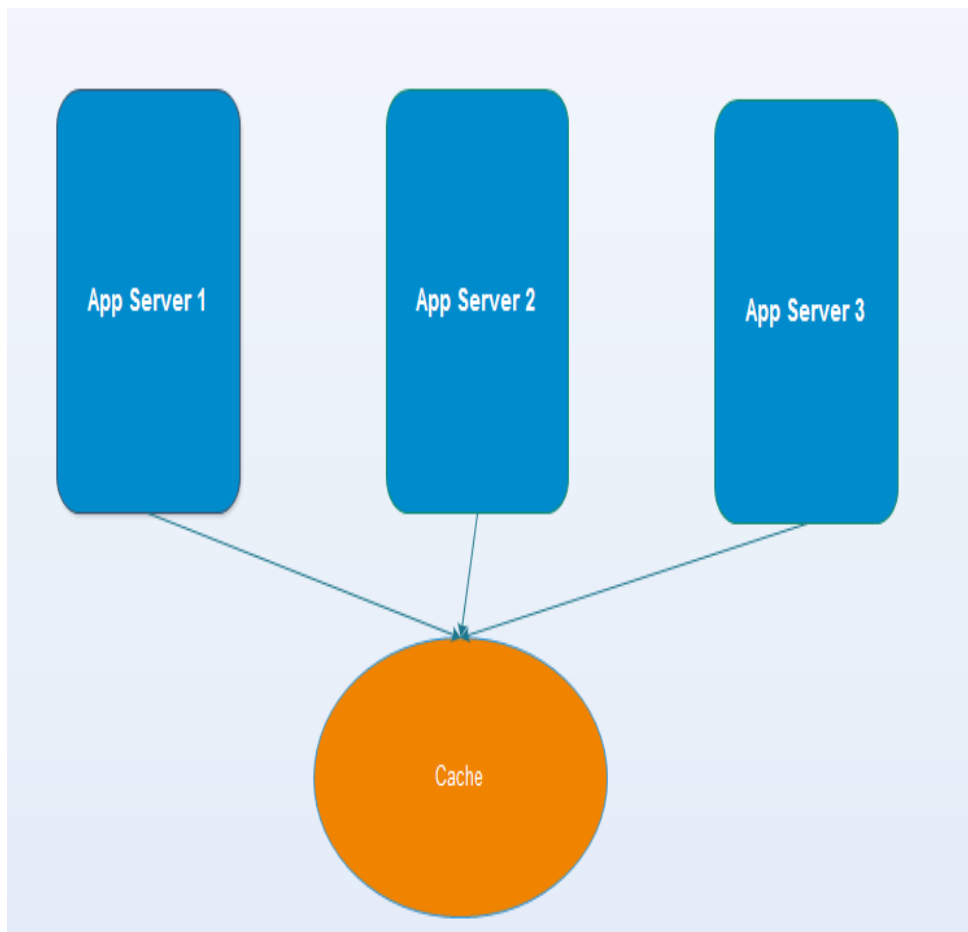
- (1) 缓存和应用在一个JVM中
- (2) 缓存间是不通信的，独立的
- (3) 弱一致性



车险理赔系统缓存2.0

Standalone(单机):

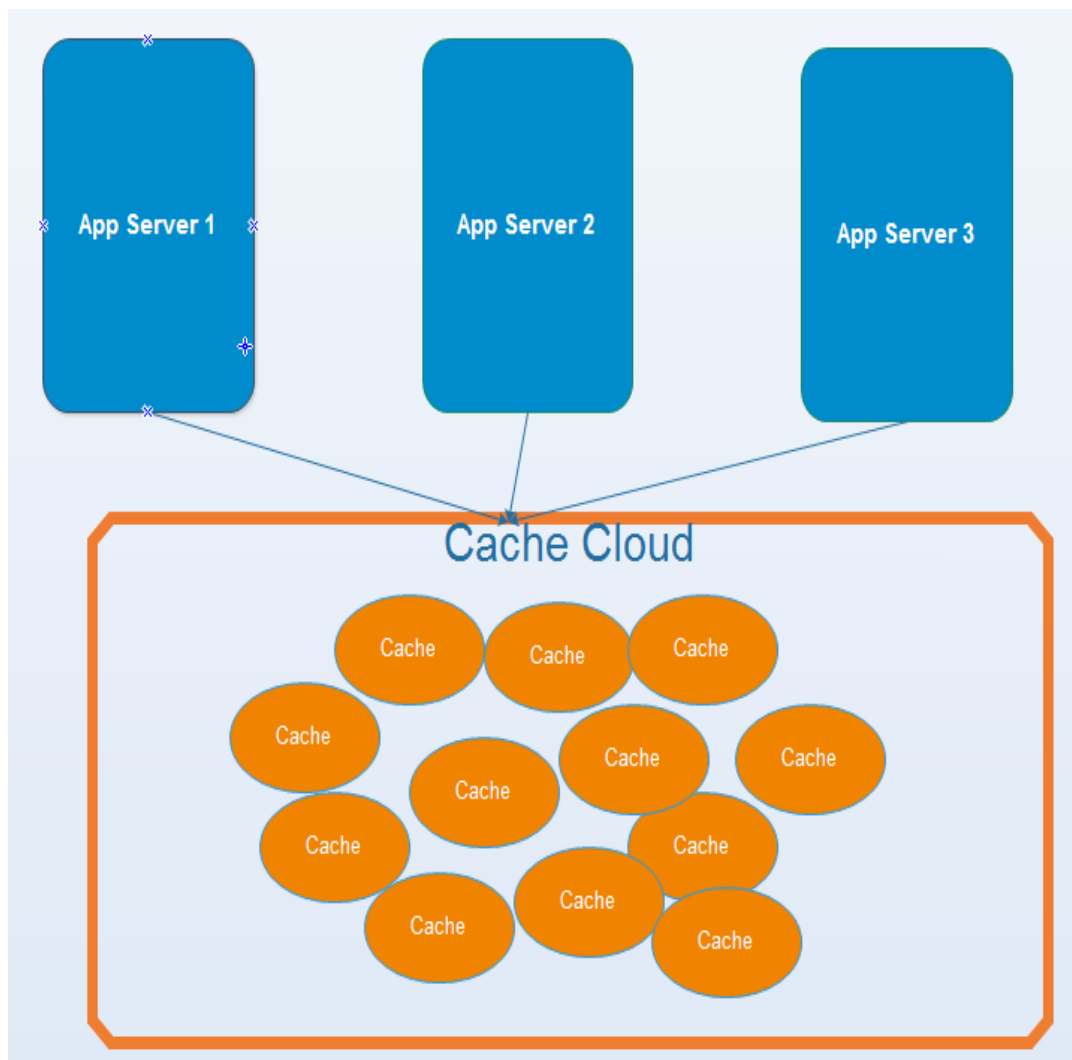
- (1) 缓存和应用是独立部署的
- (2) 缓存可以是单台
- (3) 强一致性
- (4) 跨进程、跨网络
- (5) 无高可用、无分布式



车险理赔系统缓存3.0

Distributed(分布式)

- (1) 缓存和应用是独立部署的
- (2) 多个实例
- (3) 强一致性或者最终一致性
- (4) 支持Scale Out、高可用
- (5) 跨进程、跨网络





二、Redis基础知识简介

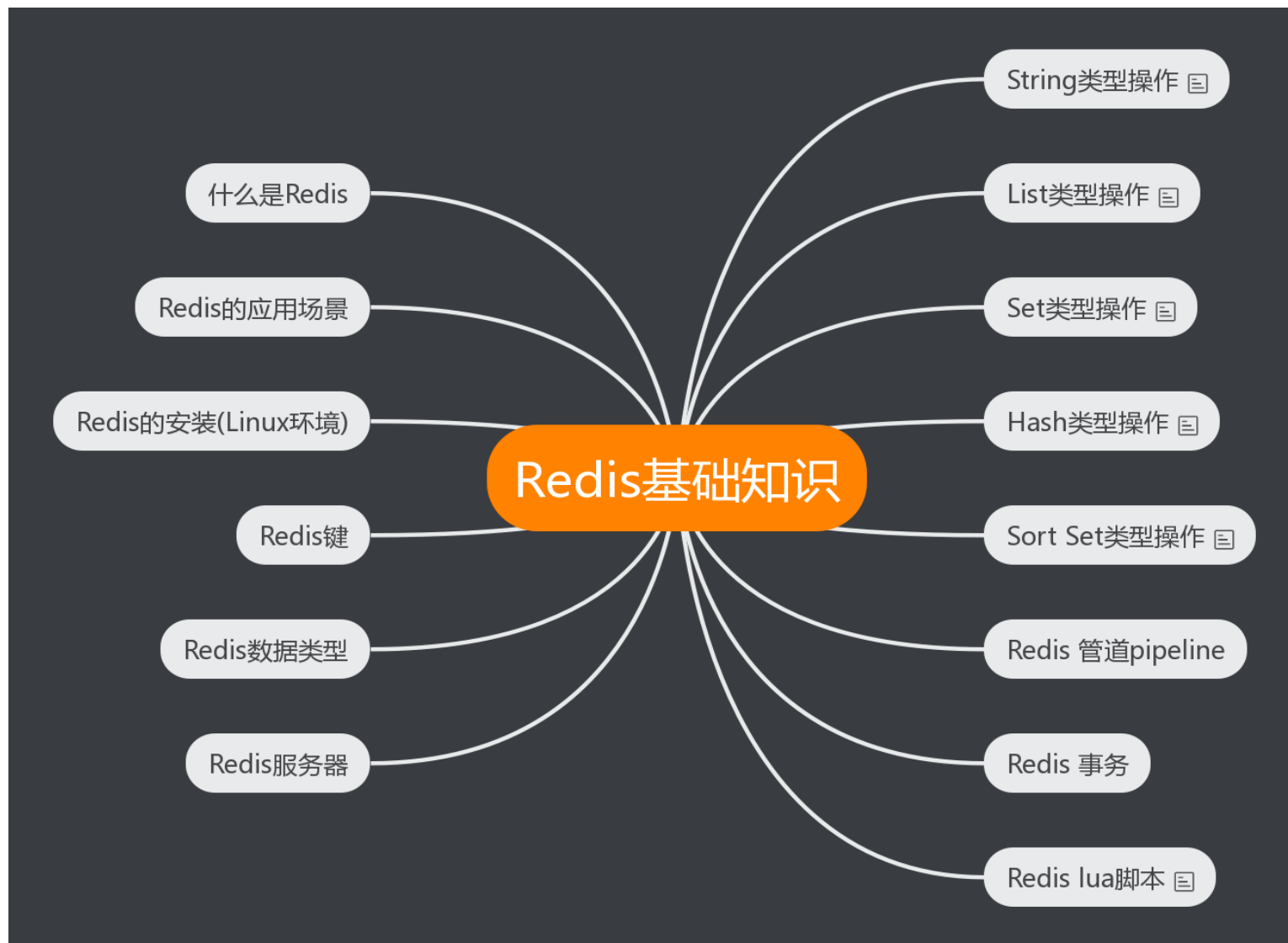
简介

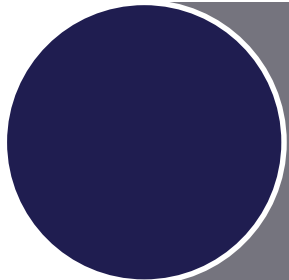
1、Redis概述

2、Redis数据类型操作

3、管道、事务、LUA脚本

看图: <https://www.processon.com/diagrams>

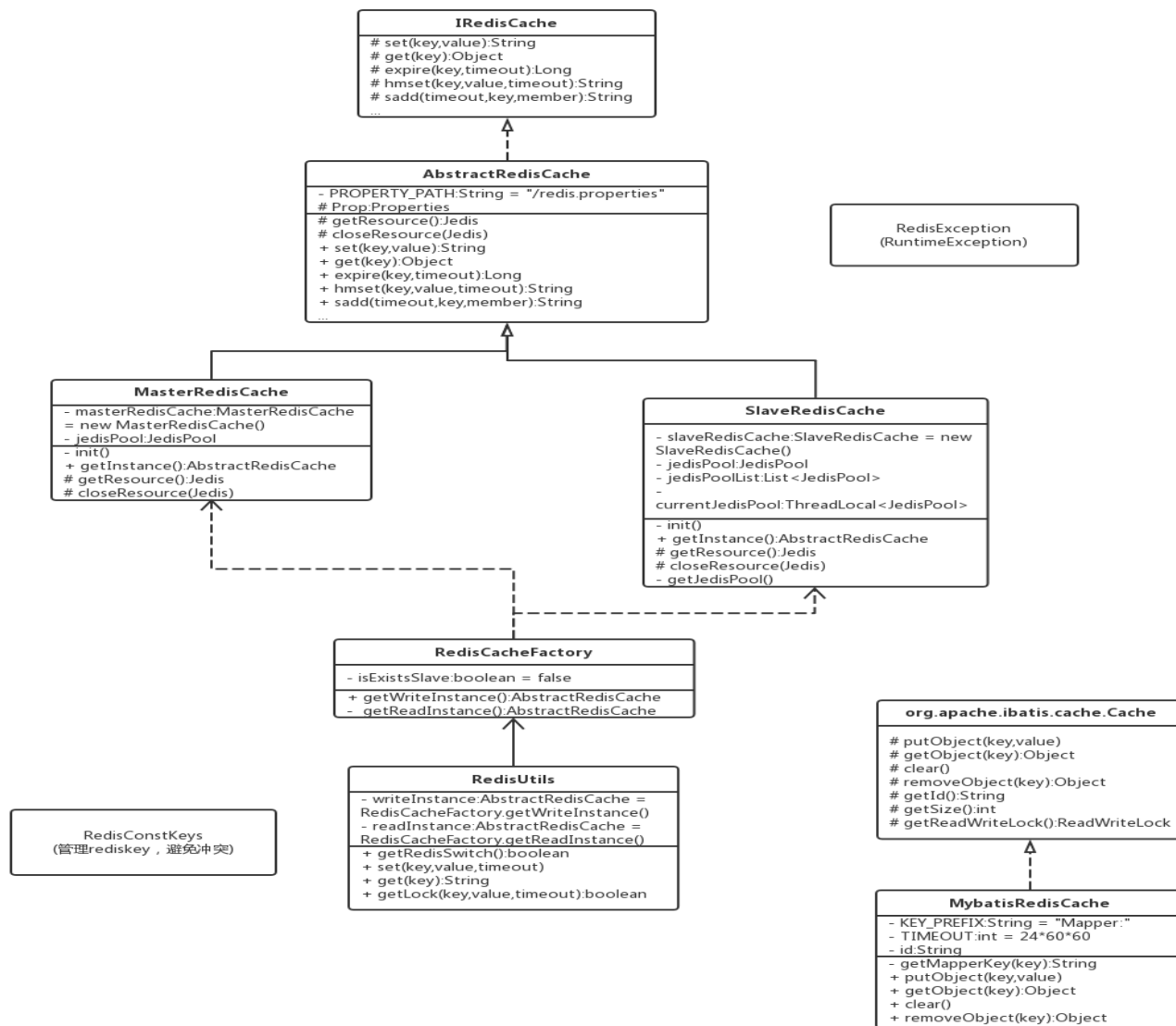




三、Redis客户端封装




看图: <https://www.processon.com/diagrams>






四、缓存常见问题分析



缓存穿透



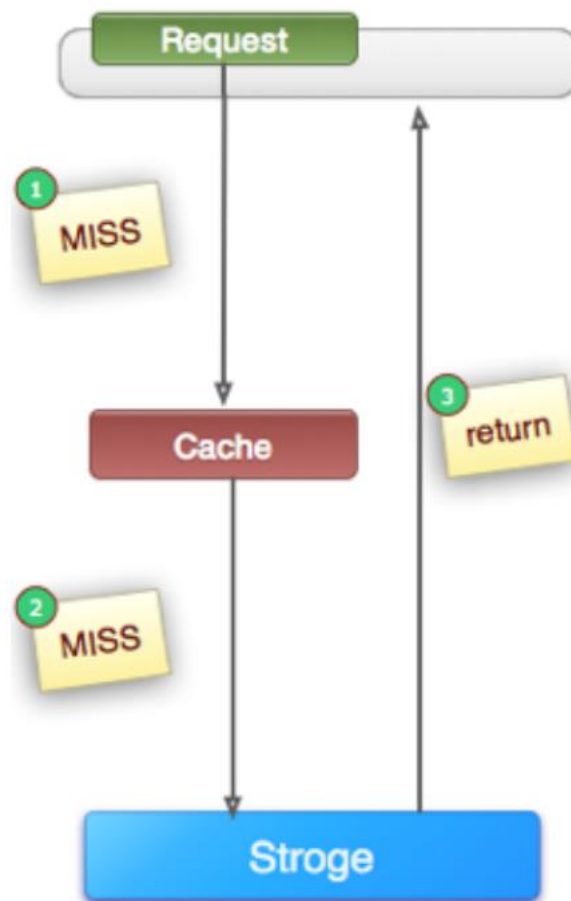
缓存雪崩



客户端异常

4.1、缓存穿透（请求数据缓存大量不命中）

- 缓存穿透是指查询一个一定不存在的数据，由于缓存不命中，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。
- 右图是一个比较典型的cache-storage架构，cache(例如redis, memcache等等) + storage(例如Oracle, mysql等等)架构，查一个压根就不存在的值, 如果不做兼容，永远会查询storage



产生原因

- 产生原因：可能是代码本身或者数据存在的问题造成的，也有可能是一些恶意攻击、爬虫等等

危害：

- 对底层数据源(oracle, mysql, http接口, rpc调用等等)压力过大，有些底层数据源不具备高并发性
- 例如mysql一般来说单台能够扛1000-QPS就已经很不错了（别说你的查询都是select * from table where id=xx 以及你的机器多么牛逼，那就有点矫情了）
- 第三方提供的一个抗压性很差的http接口，可能穿透会击溃他的服务

解决方法

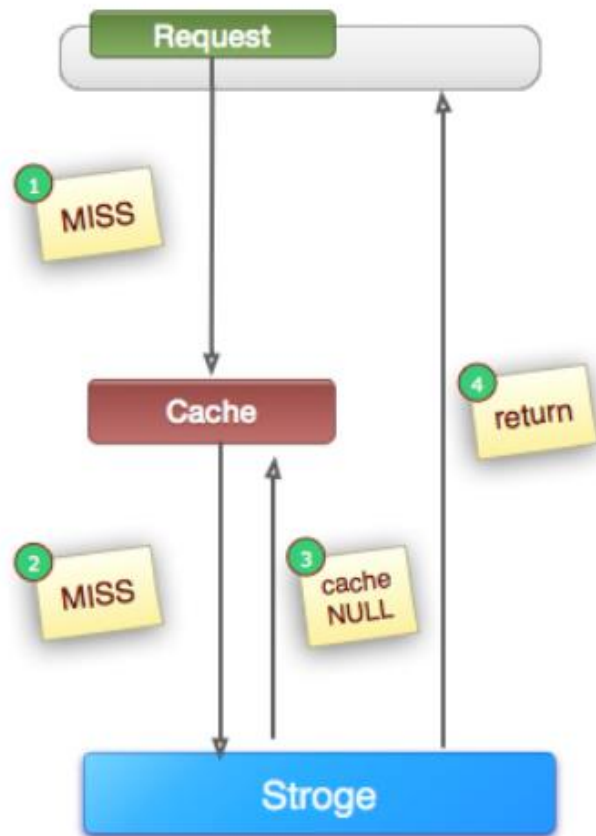
缓存空对象：

- 如右图所示，当第②步MISS后，仍然将空对象保留到Cache中（可能是保留几分钟或者一段时间，具体问题具体分析），下次新的Request（同一个key）将会从Cache中获取到数据，保护了后端的Storage。

- 维护成本：低，但是有两个问题：

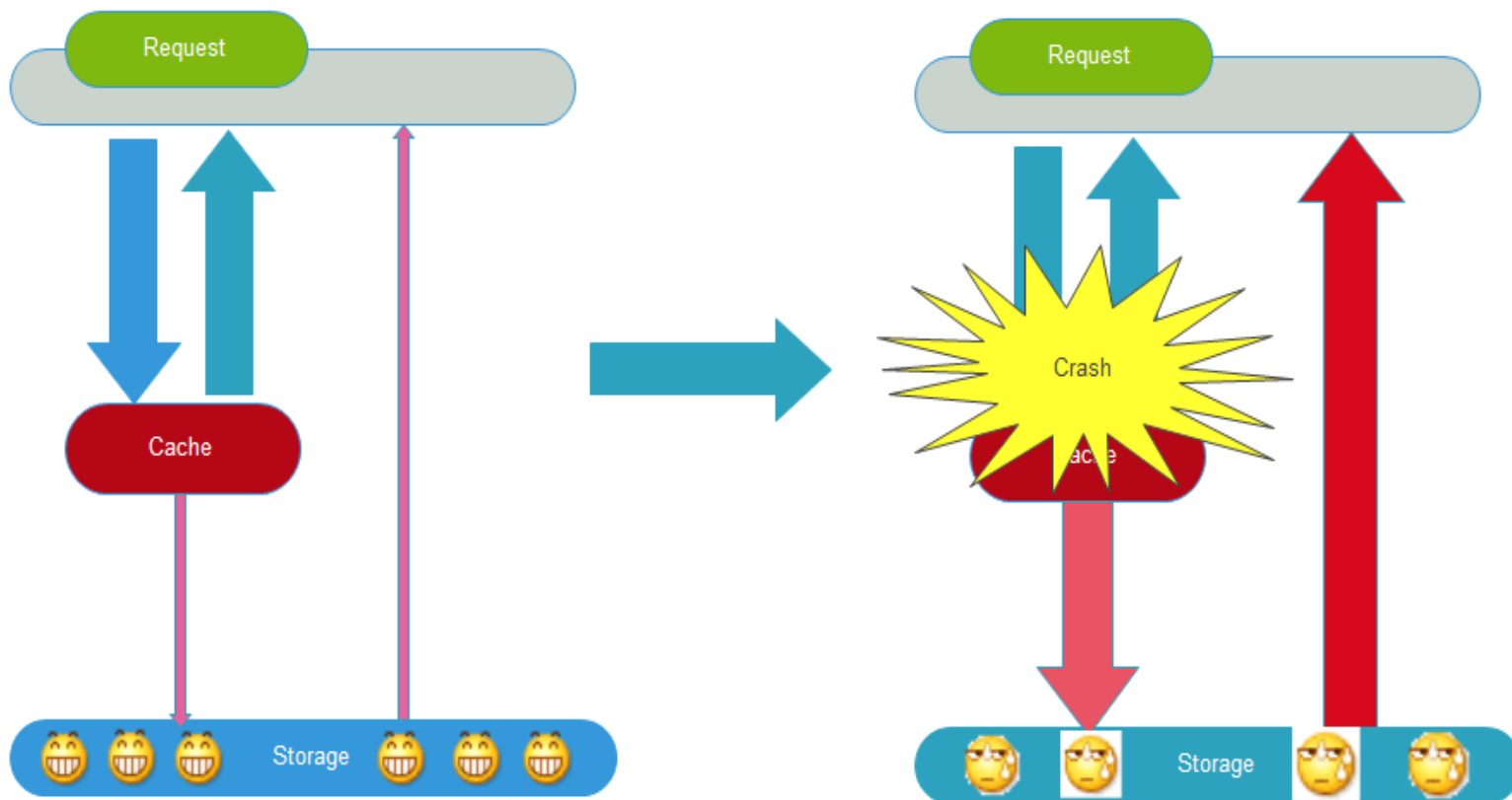
1、空值做了缓存，意味着缓存系统中存了更多的key-value，也就是需要更多空间（有人说空值没多少，但是架不住多啊），解决方法是我们可以设置一个较短的过期时间。

2、数据会有一段时间窗口的不一致，假如，Cache设置了5分钟过期，此时Storage确实有了这个数据的值，那这段时间就会出现数据不一致，解决方法是我们可以利用消息或者其他方式，清除掉Cache中的数据。



4.2、缓存雪崩

1. 由于Cache层承载着大量请求，有效的保护了Storage层(通常认为此层抗压能力稍弱)，所以Storage的调用量实际很低
2. 如果Cache层由于某些原因(宕机、cache服务挂了或者不响应了)整体crash掉了，也就意味着所有的请求都会达到Storage层，所有Storage的调用量会暴增，所以它有点扛不住，甚至可能会挂掉

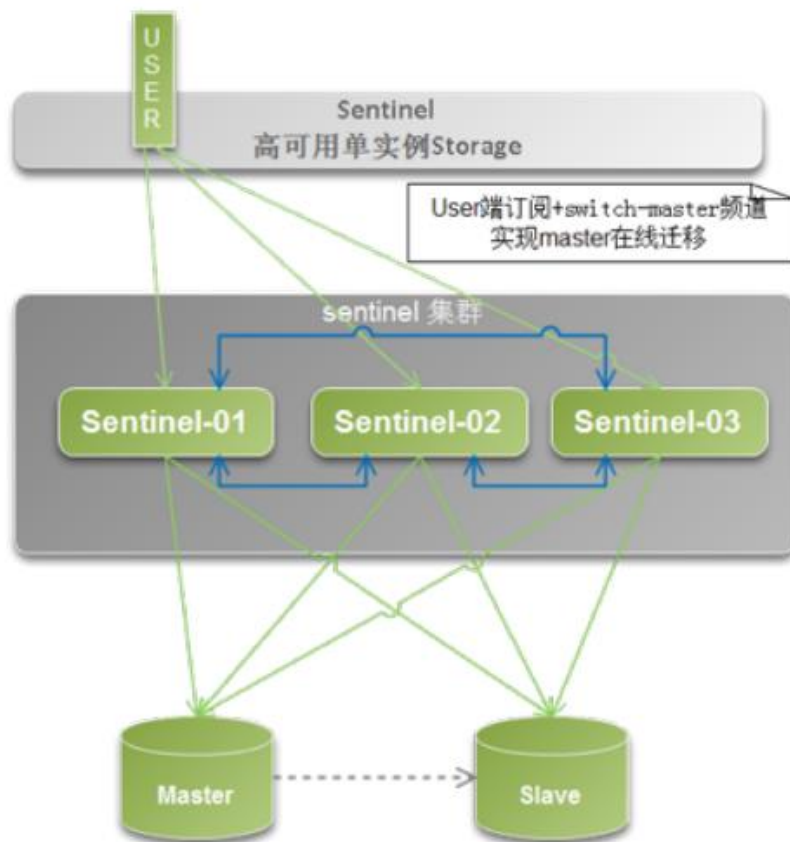
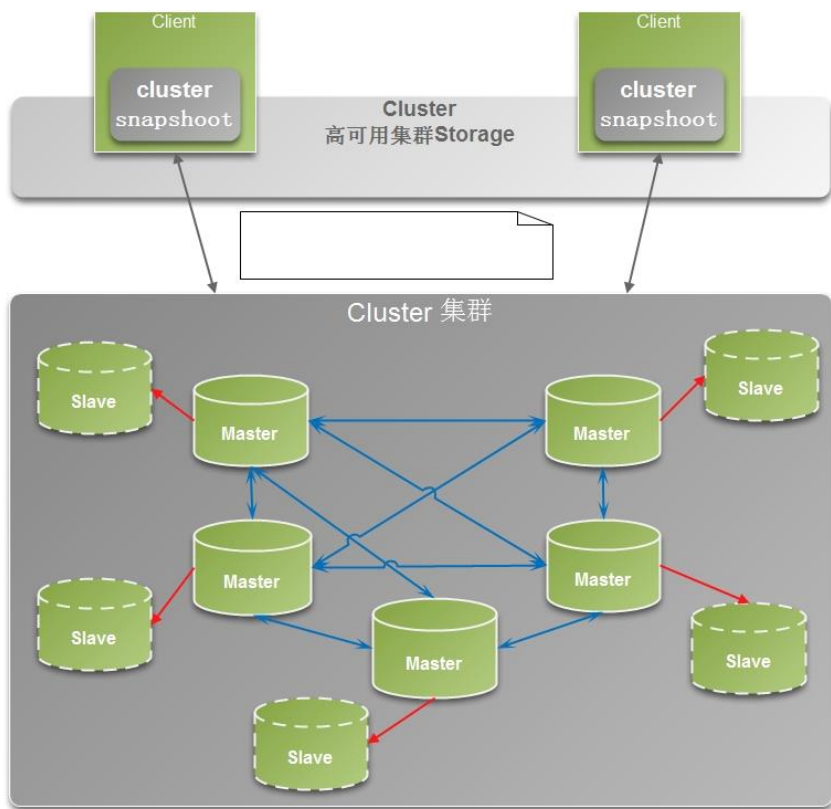


危害:

雪崩的危害显而易见，通常来讲可能很久以前storage已经扛不住大量请求了，于是加了cache层，所以雪崩会使得storage压力山大，甚至是挂掉。

如何预防:

保证Cache服务高可用性，redis的cluster 和sentinel机制，服务降级。



4.3、客户端异常

1).无法从连接池获取到连接

异常信息：

redis.clients.jedis.exceptions.JedisConnectionException: Could not get a resource from the pool

分析原因：

- 客户端：高并发下连接池设置过小，出现供不应求。
- 客户端：没有正确使用连接池，比如没有进行释放。
- 客户端：存在慢查询操作，这些慢查询持有的Jedis对象归还速度会比较慢，造成池子满了。
- 服务端：客户端是正常的，但是Redis服务端由于一些原因造成了客户端命令执行过程的阻塞

2). 客户端读写超时

异常信息:

```
redis.clients.jedis.exceptions.JedisConnectionException:  
java.net.SocketTimeoutException: Read timed out
```

分析原因:

- 读写超时设置的过短。
- 命令本身就比较慢。
- 客户端与服务端网络不正常。
- Redis自身发生阻塞。

3). 客户端连接超时

异常信息:

```
redis.clients.jedis.exceptions.JedisConnectionException:  
java.net.SocketTimeoutException: connect timed out
```

分析原因:

- 连接超时设置的过短。
- Redis发生阻塞，造成tcp-backlog已满，造成新的连接失败。
- 客户端与服务端网络不正常。

4). 客户端缓冲区异常

异常信息:

redis.clients.jedis.exceptions.JedisConnectionException: Unexpected end of stream.

分析原因:

- 输出缓冲区满。config set client-output-buffer-limit "normal 1048576 1048576 60"
- 长时间闲置连接被服务端主动断开，可以查询timeout配置的设置以及自身连接池配置是否需要做空闲检测。
- 不正常并发读写：Jedis对象同时被多个线程并发操作，可能会出现上述异常。

5). 其他可能引发异常情况

- Lua脚本正在执行
- Redis正在加载持久化文件
- Redis使用的内存超过maxmemory配置
-