新浪微博
weibo.com

分布式缓存架构基础

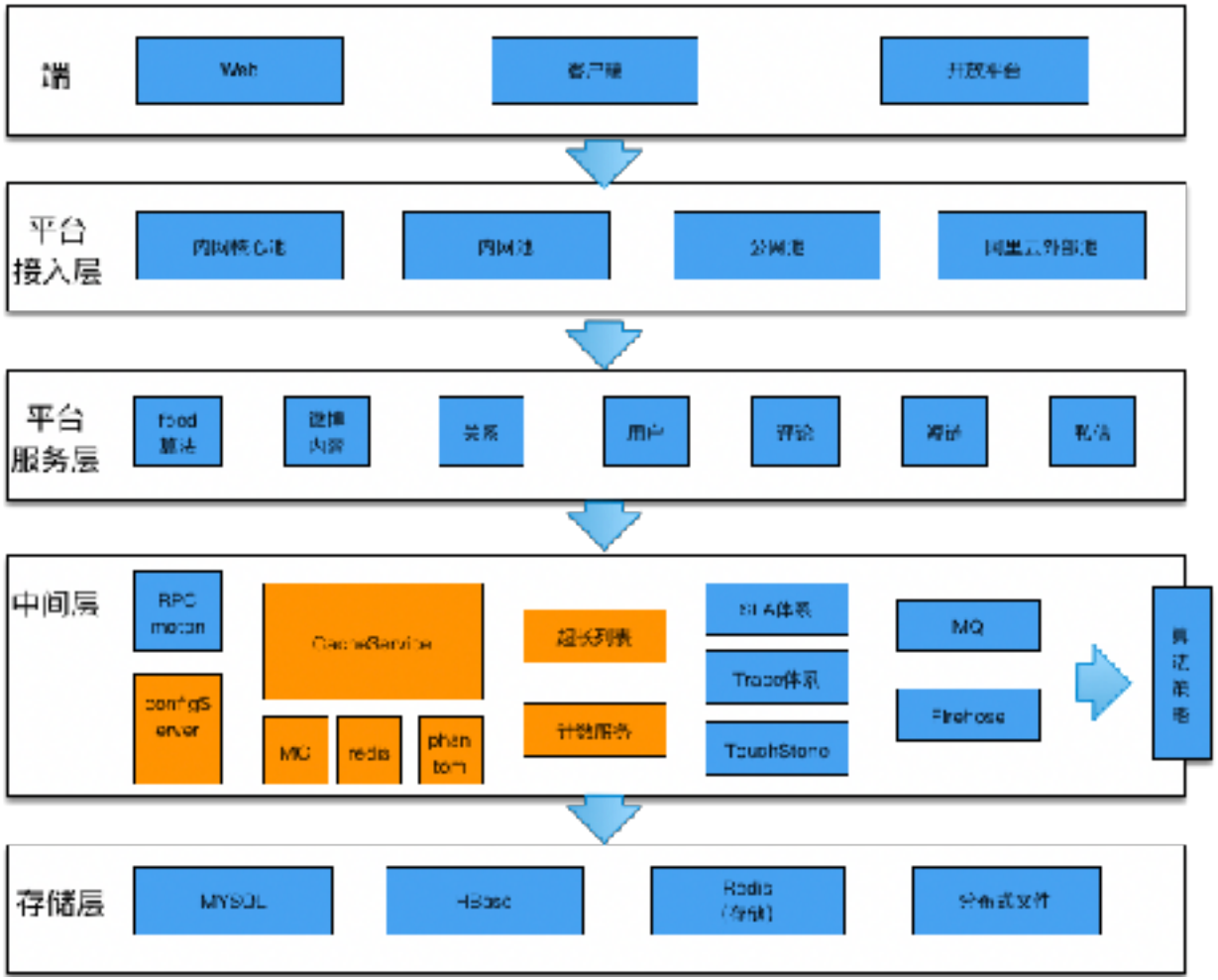
刘东辉

@Alfejik

大纲

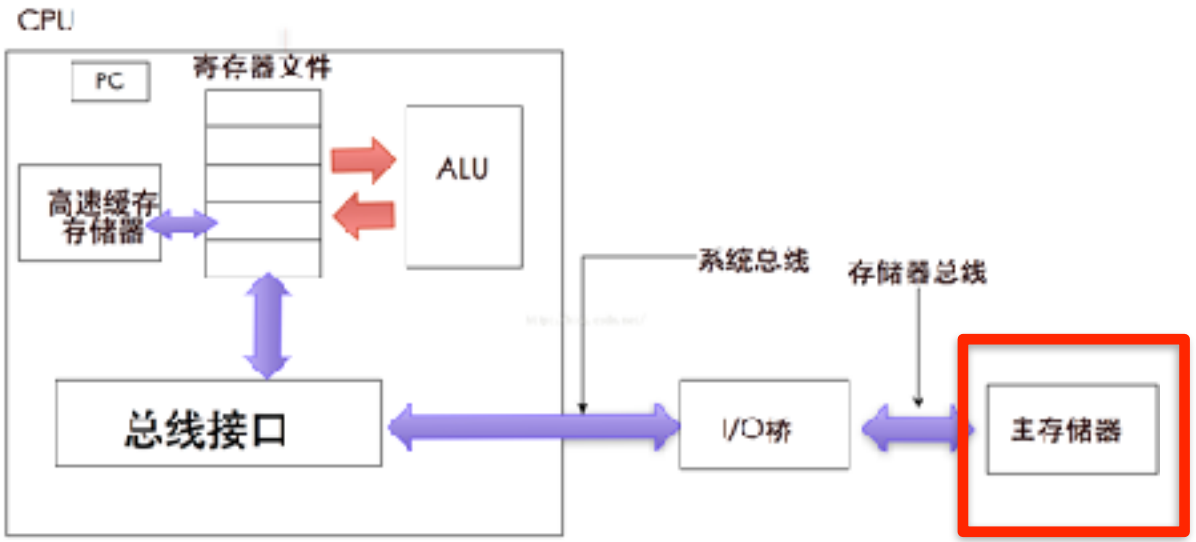
- 缓存概述
- 常用缓存服务
- 分布式缓存实现
- 缓存服务化
- 缓存实践案例

- 缓存在微博架构的位置


端
Web
平台
接入层
平台
服务层
中间层
RPC
memcn
configS
erver
MC
redis
phen
tom
MQ
Trace体系
TouchStore
Redis
存储层
MYSQL
Hbase

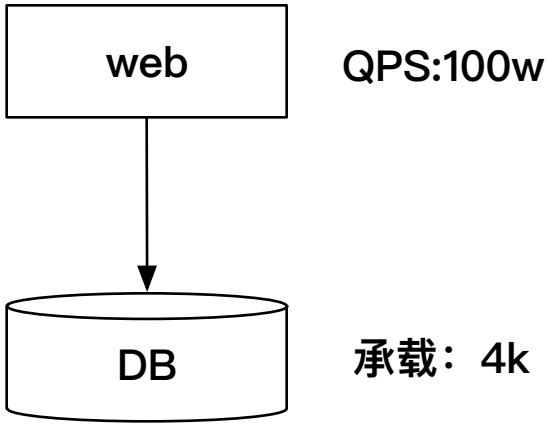
缓存概述

- 什么是缓存
  - 存储数据
  - 加快数据存取性能



高速缓存存储器

- 为什么引入缓存
  - 单一DB存储结构的问题
    - 性能提升有限，难以量级上的提升
    - 成本高昂



新浪微博 weibo.com

- 各类存储介质的访问速度

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

1. 内存比磁盘速度快

2. 数量级提升

- 线上性能评估
  - 内存型
    - redis: rw 6w/s
    - memcached: rw 20w/s
  - 磁盘型
    - mysql(ssd): read 2k/s, write: 4k/s

软件类型	机型	单机可用内存	单机可用磁盘	qps1	说明1	qps2	说明2	单机极限带宽	备注	评估公式
redis	M45	100G		6w	正常	3w	hgetall	800mb	特殊情景需要特殊评估	带宽和qps按照20%冗余计算，取最大台数
mc	M45	100G		20w	正常			800mb	特殊情景需要特殊评估	带宽和qps按照50%冗余计算，取最大台数
mcq	M45		500G	8c	1k value	4c	4k value	800mb		qps按照50%冗余计算台数
HBase	V41		27T	1w	write	2w	read			qps按照50%冗余计算，和容量比对取最大台数
hadoop	S41		27T							按照可用容量计算台数
MySQL	D46		3T	2c	write	4c	read		特殊情景需要特殊评估	qps按照50%冗余计算，和容量比对取最大台数
Kafka	M45		500G	300mb	正常					qps按照50%冗余计算台数

缓存概述

- 缓存价格

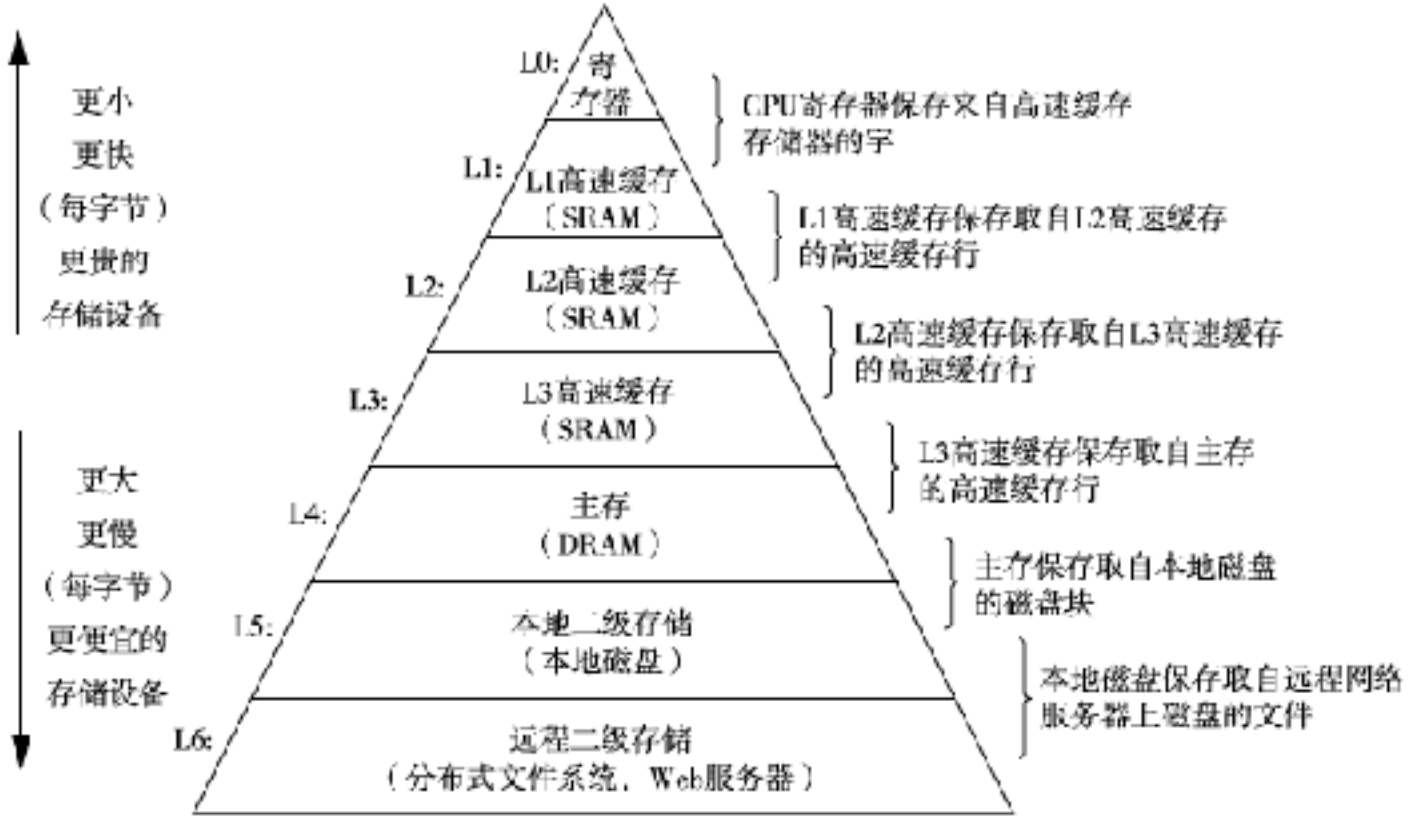
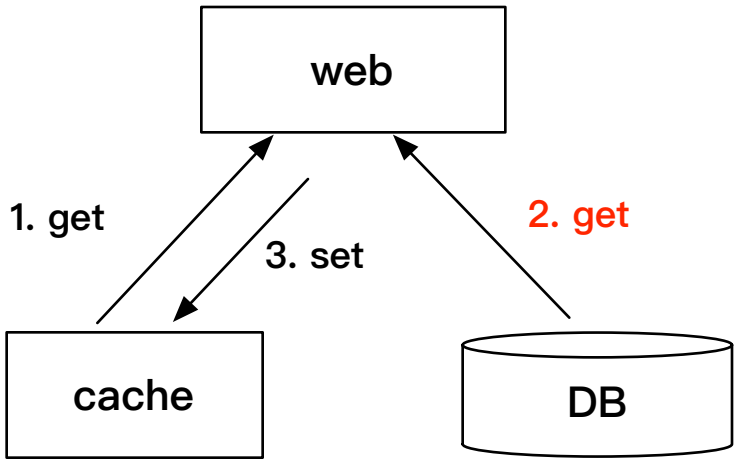


图 1 9　一个存储器层次结构的示例

性能 vs 成本：不能把所有的数据放入内存

缓存概述

- 缓存适用场景
  - 业务请求量大、性能要求高
  - 业务数据冷热区分明显
    - 热数据(频繁访问)放入内存



常用缓存服务

- 缓存分类
  - Local Cache(应用程序内)
    - Map & Ehcache(java第三方库)
  - 缓存组件(独立部署)
    - Memcached
    - Redis
    - counterservice
    - phantom
    - pika

常用缓存服务

- Local Cache适用场景
  - 响应时间敏感
  - 热点数据量小，但访问量巨大
  - 业务能接受一定程度数据的不一致性

常用缓存服务

- **Memcached、Redis介绍**

	Memcached	Redis
组件类型	kv存储	kv存储
数据存储	内存	内存
数据类型	简单kv	list、hash、set等
内存管理	slab	jemalloc等
请求处理	多线程	单线程
过期	被动+lru crawler线程	主动+被动
LRU	局部lru	allkeys_lru、volatile_lru等
持久化	不支持	支持
主从复制	不支持	支持
集群	client端实现	>3.0版本支持

常用缓存服务

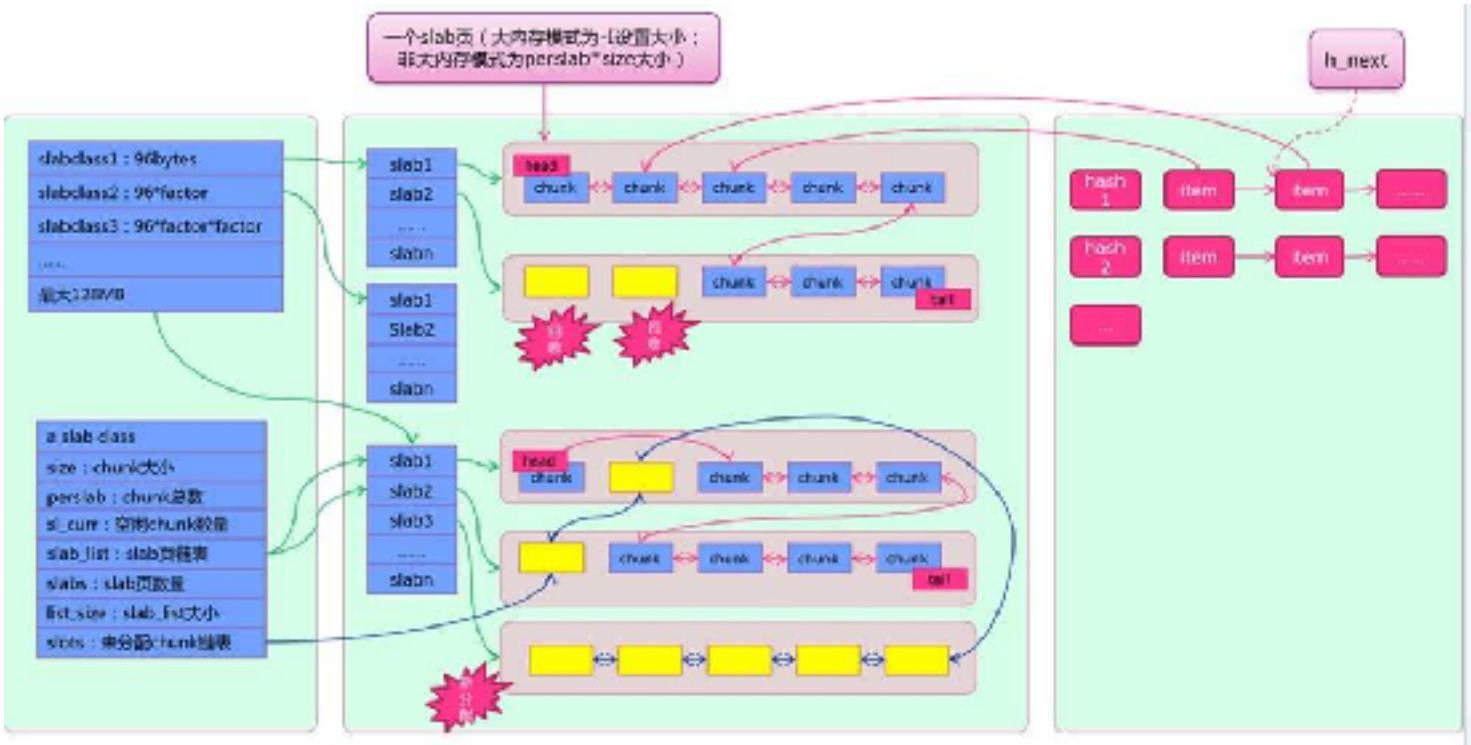
新浪微博 weibo.com

- Memcached & Redis适用场景
  - Memcached
    - 简单kv
  - Redis
    - 复杂数据类型
    - 复杂运算
    - 内存存储

新浪微博 weibo.com

- Memcached内存管理
  - chunk
    - 用于存储记录的内存块
  - slabclass
    - 特定大小的chunk的组
  - slab(page)
    - 分配给slab class的内存(默认1MB)
    - 分配给slab class之后根据slab class的大小切分成chunk
    - slab分配给指定slabclass后，会被该slabclass一直占用

- **Memcached内存管理**



常用缓存服务

新浪微博 weibo.com

- Memcached淘汰策略
  - LRU(最近最久未使用)
  - slabclass内部LRU

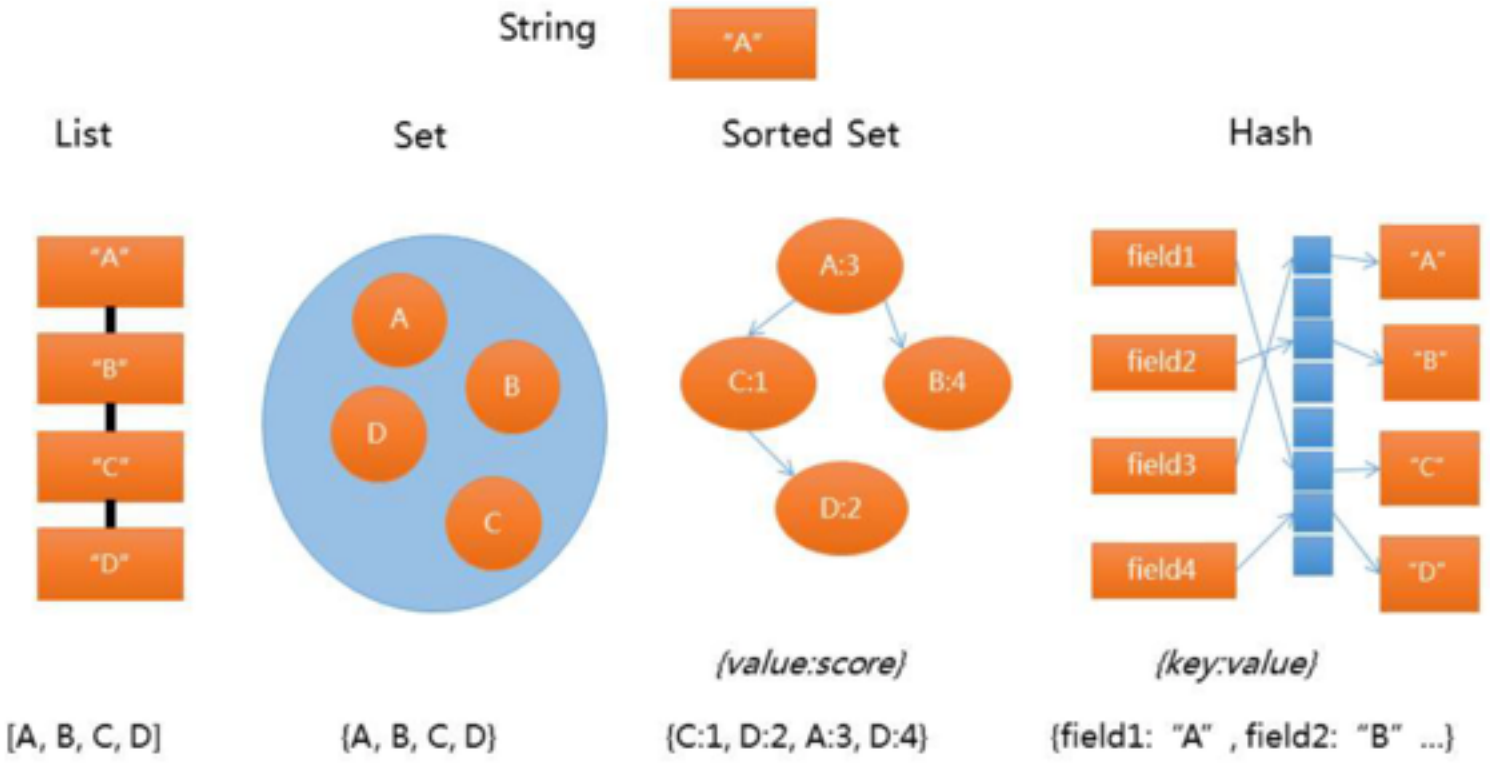
常用缓存介绍

- Memcached练习
  - 测试实例: 10.210.230.36:10001
  - 命令： set/get, add/replace， cas/gets, stats

```
[ldh@centos-linux src]$ telnet 10.210.230.36 10001
Trying 10.210.230.36...
Connected to 10.210.230.36.
Escape character is '^]'.
set hello 32 0 5
world
STORED
get hello
VALUE hello 32 5
world
END
```

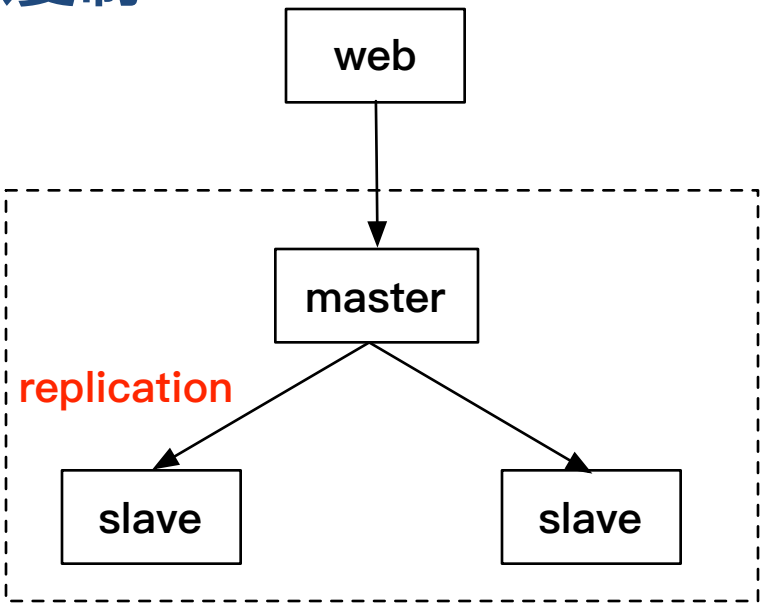
常用缓存服务

- **Redis多数据类型**



常用缓存服务

- Redis持久化机制
  - 内存快照: rdb
  - 追加日志：aof
- Redis主从复制



新浪微博 weibo.com

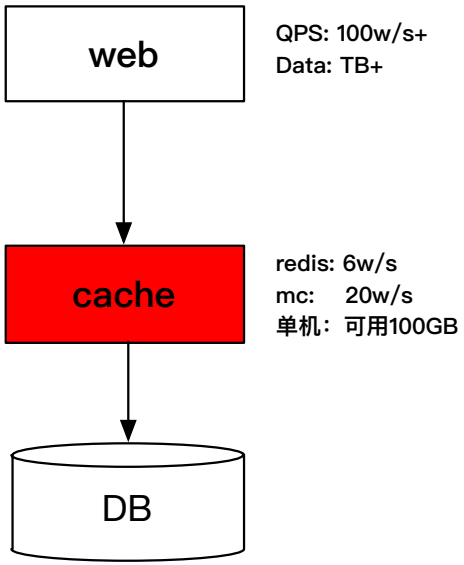
- Redis练习
  - 测试实例: master: 10.210.230.36:7777

    slave: 10.210.230.36:7778
  - 命令： set/get, hset/hget, lpush/lpop, sadd

```
[ldh@centos-linux src]$ redis-cli -h 10.210.230.36 -p 7777
10.210.230.36:7777> set hello world
OK
10.210.230.36:7777> get hello
"world"
10.210.230.36:7777> hset book name "redis in action"
(integer) 0
10.210.230.36:7777> hget book name
"redis in action"
10.210.230.36:7777> sadd fruit apple pear watermelon
(integer) 3
10.210.230.36:7777> smembers fruit
1) "pear"
2) "watermelon"
3) "apple"
```

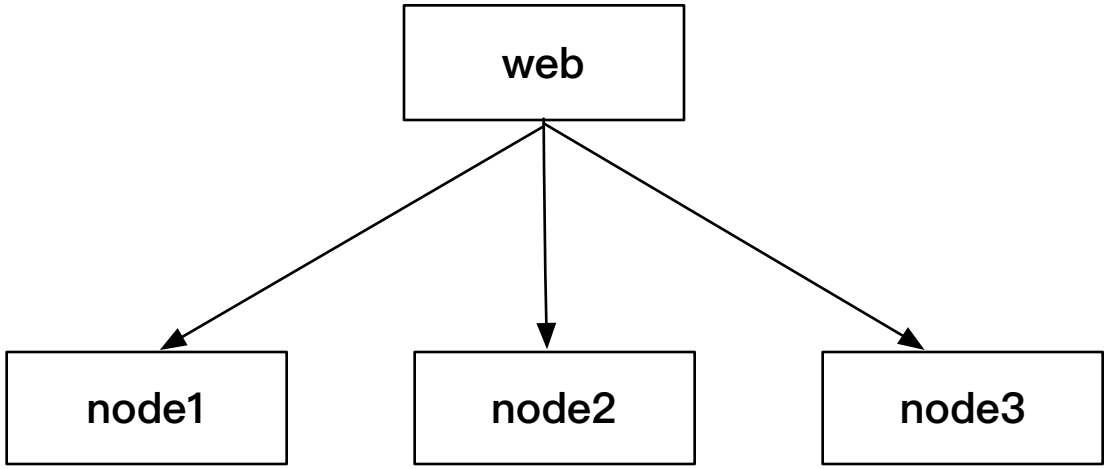
分布式缓存实现

- 为什么需要分布式缓存
  - 单实例缓存架构
    - 容量问题 -> 无法缓存TB级数据
    - 请求量问题 -> 无法承载百万级QPS
    - 高可用(HA)问题 -> 单节点故障
    - 扩展问题 -> 无法平滑扩展



分布式缓存实现

- 数据分片
  - 单个实例 -> 一组实例
  - 数据分散
  - 请求量分散

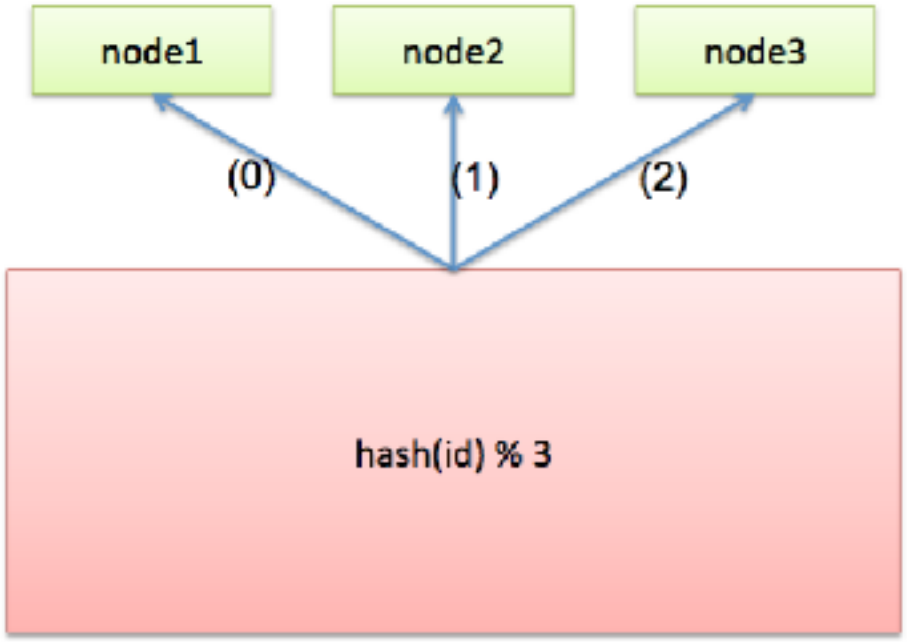


分布式缓存实现

- 数据分片常用的算法
  - 取模求余
  - 一致性哈希

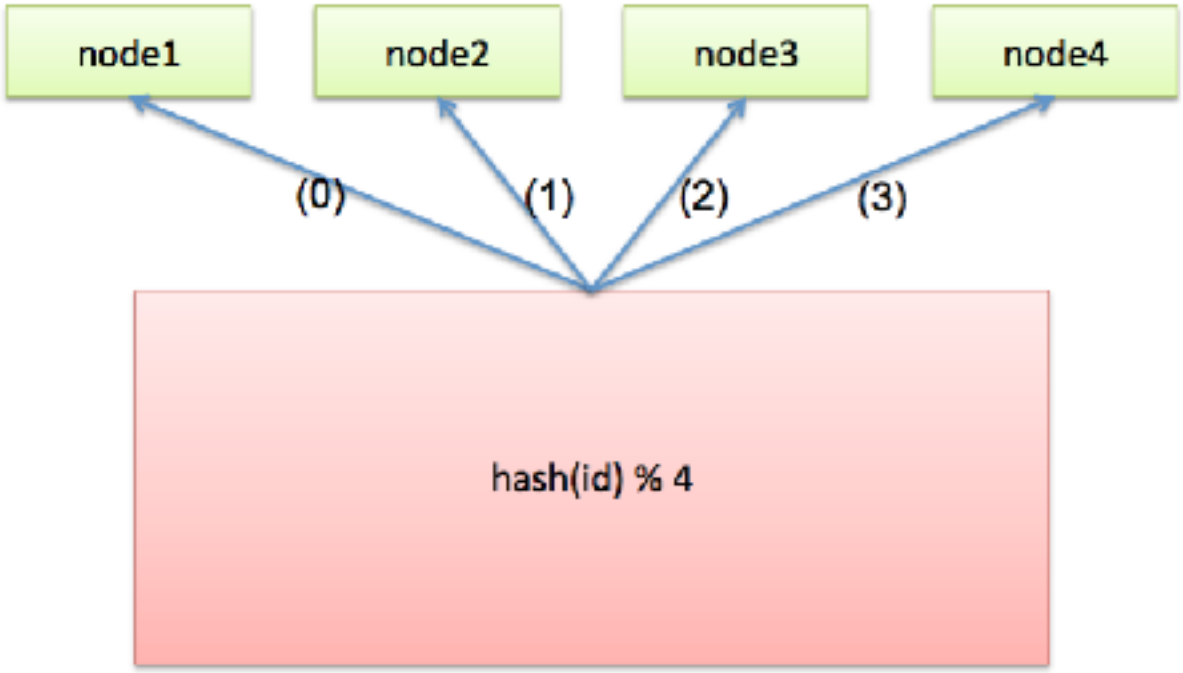
分布式缓存实现

- 取模求余算法

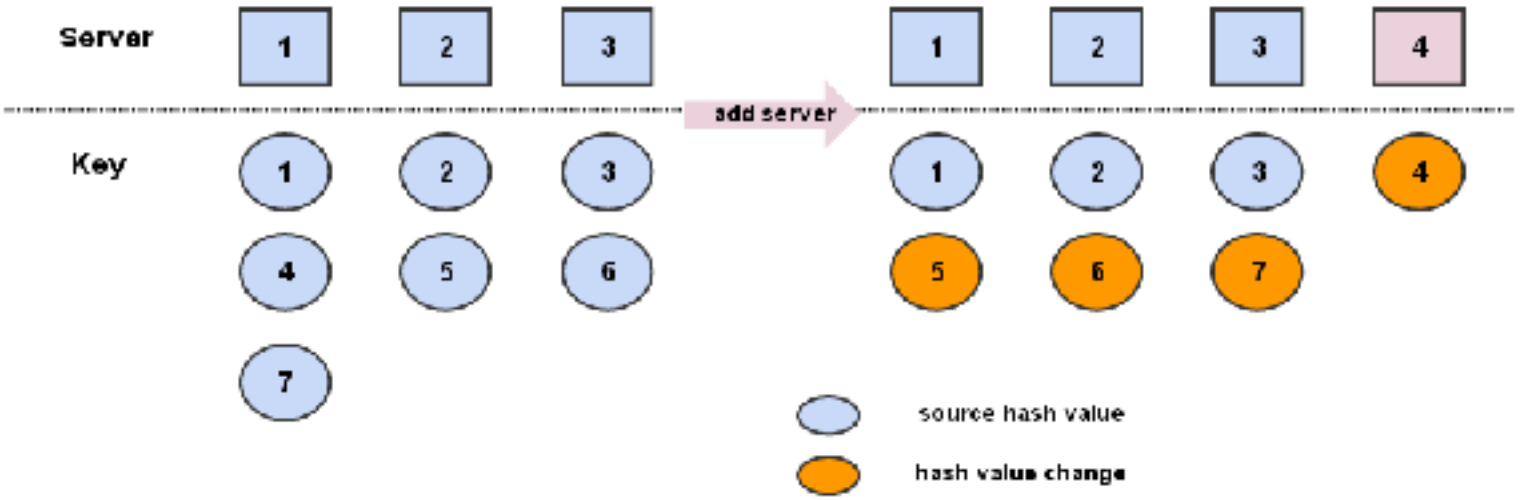


分布式缓存实现

- 取模求余算法 - 加节点

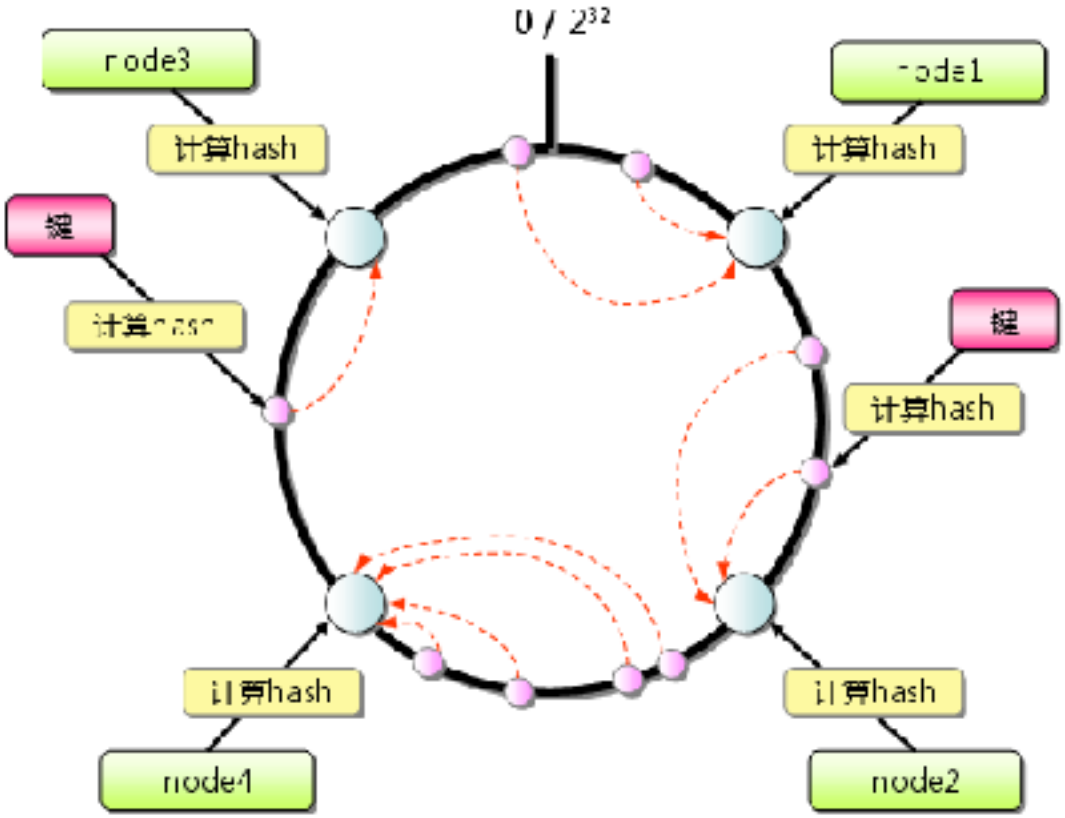


- 取模求余算法 - 加节点


Server
1
2
3
1
2
3
4
add server
Key
1
2
3
4
5
6
7
1
2
3
4
5
6
7
source hash value
hash value change

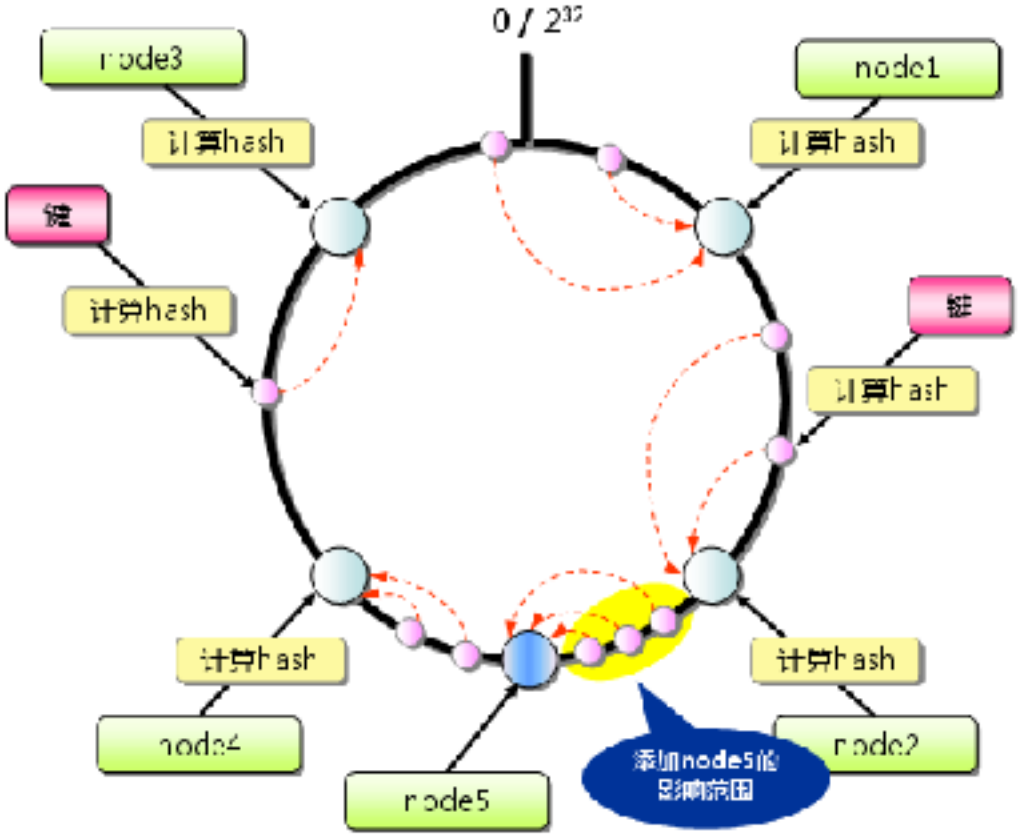
分布式缓存实现

- 一致性哈希算法



分布式缓存实现

- 一致性哈希算法 - 加节点

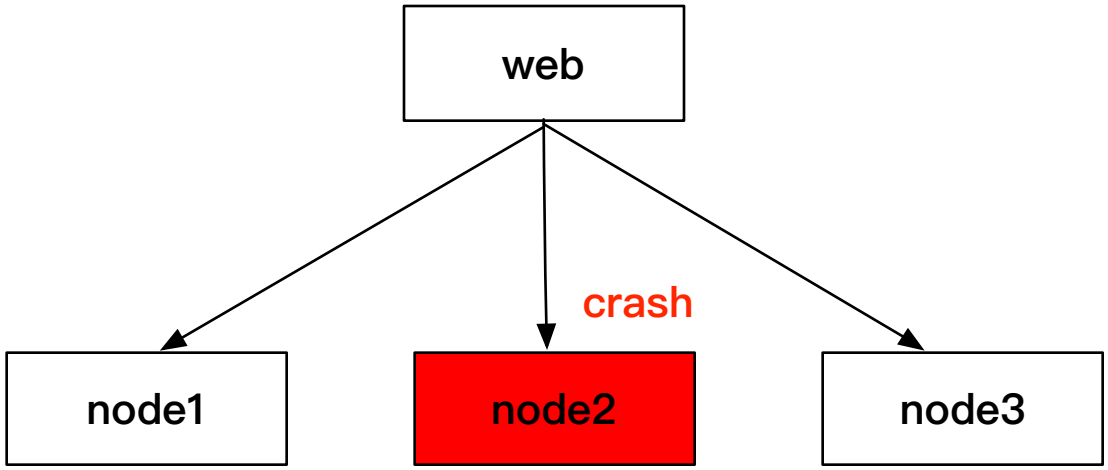


分布式缓存实现

- 取模求余
  - 优点：算法简单
  - 缺点：加减节点时震荡厉害，命中率下降厉害
- 一致性哈希
  - 优点：加减节点震荡较小，保证较高的命中率
  - 缺点：负载不够均匀， 出现热节点

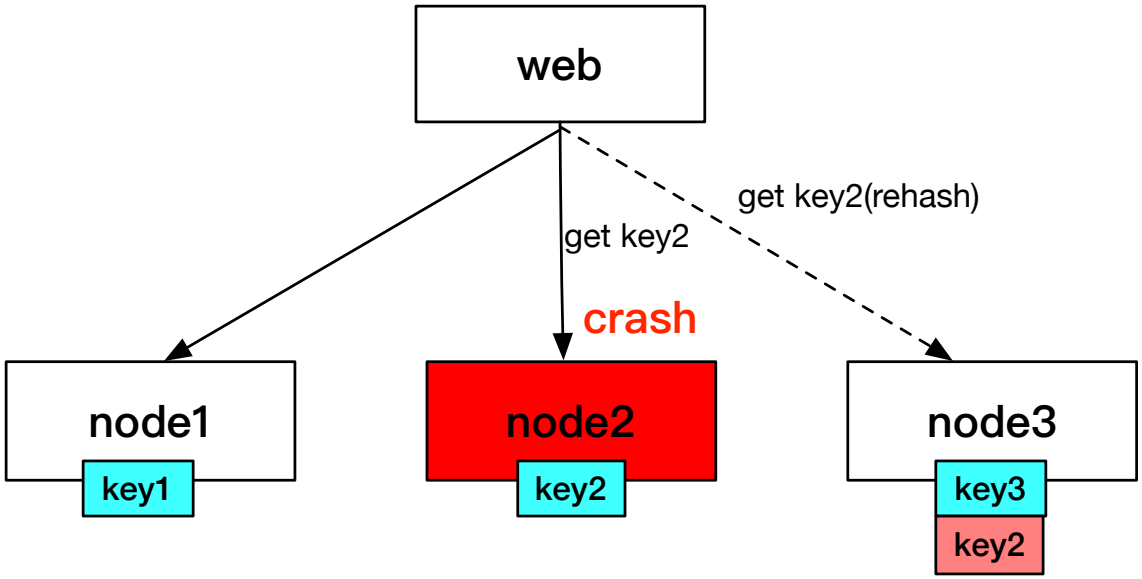
分布式缓存实现

- 单层结构问题
  - 由于机器宕机等原因，某一个缓存实例不可用
    - 大量请求持续穿透到DB
    - 数据库被冲垮，业务系统出现异常



分布式缓存实现

- Rehash策略
  - hash到故障节点重新rehash
  - 保存数据到rehash后的节点避免对存储二次压力
  - 解决存储层负载居高不下问题



- Rehash策略的问题
  - 容量、请求量易成为瓶颈
  - 出现脏数据

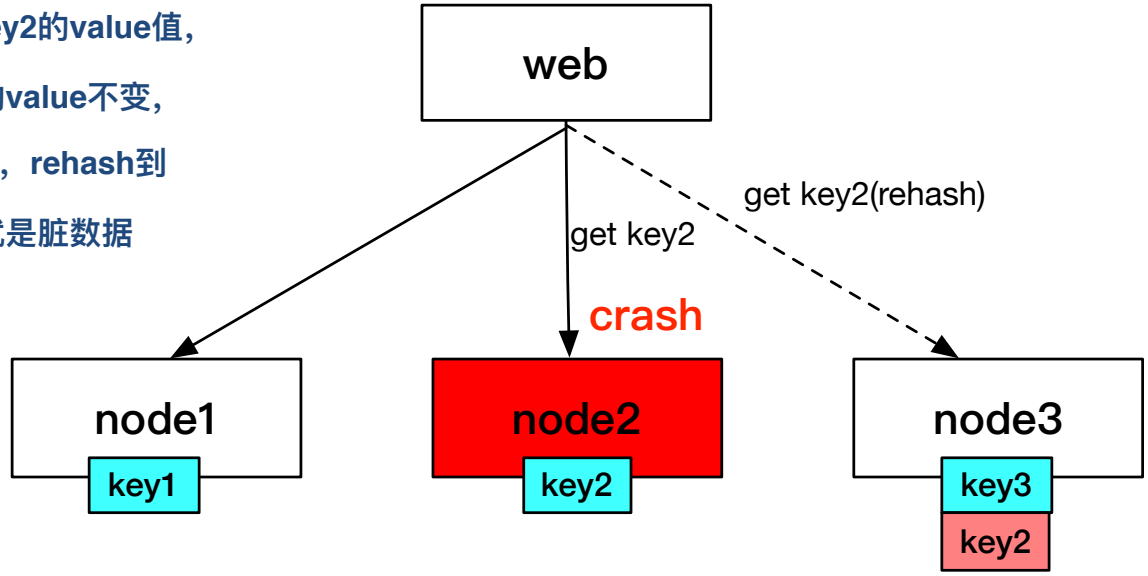
场景举例：

node2恢复后，变更key2的value值，

这时候node3中key2的value不变，

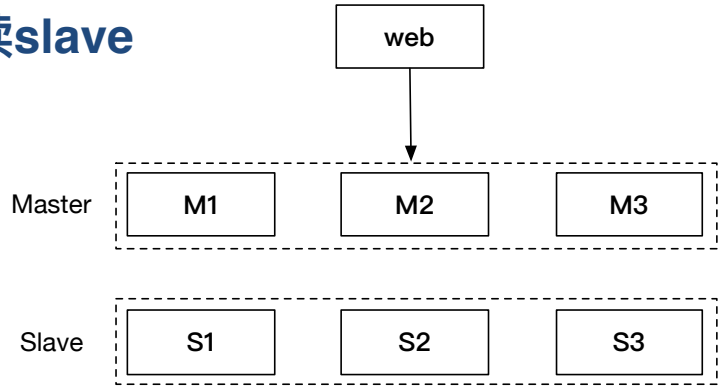
下次node2再出现问题，rehash到

node3取到的key2值就是脏数据



分布式缓存实现

- M-S双层结构
  - slave做HA备份
  - 更新逻辑
    - 双写
  - 读取逻辑
    - 先master，miss或失败后读slave
  - 一致性
    - 以master为准
    - cas master, set slave

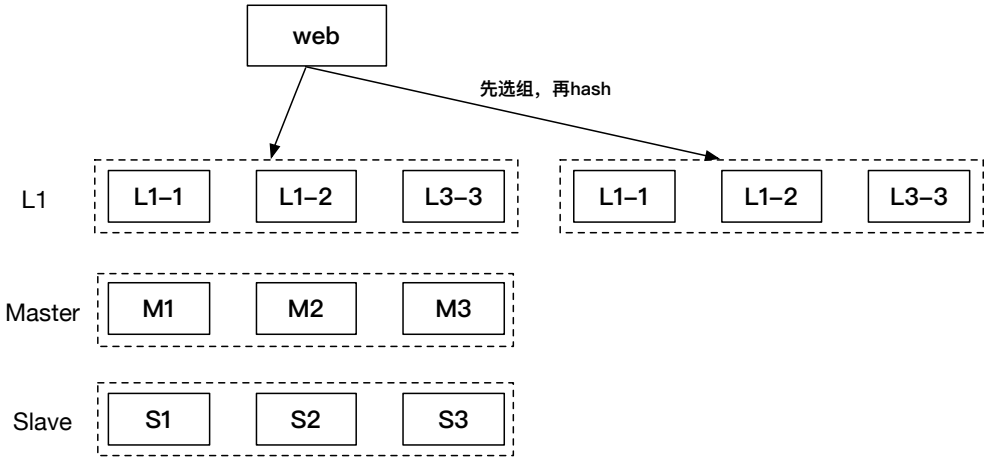


分布式缓存实现

- M-S双层结构问题
  - 数据分片后，请求量仍然很大
    - 单个分片出现带宽、cpu瓶颈
  - 应对热点事件需要快速扩展系统承载能力

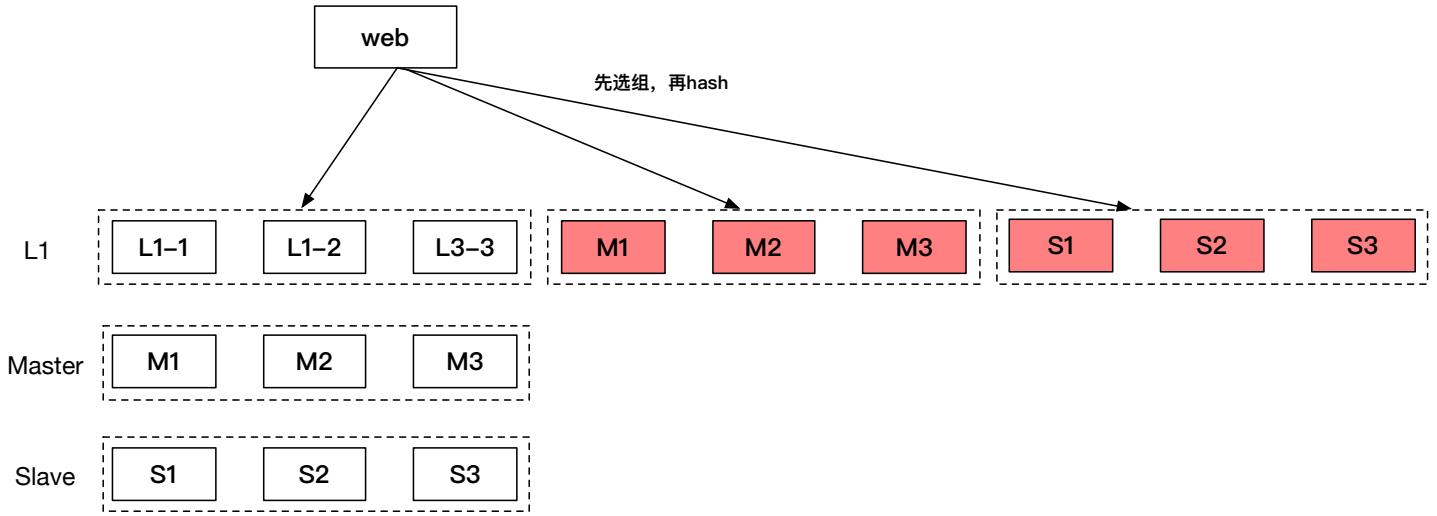
分布式缓存实现

- L1-M-S三层结构
  - 成组扩容
    - 相比master、slave一般容量小
  - 更新逻辑
    - 多写
  - 读取逻辑
    - L1 -> master -> slave
  - 一致性
    - 以master为准
    - cas master, set slave,L1

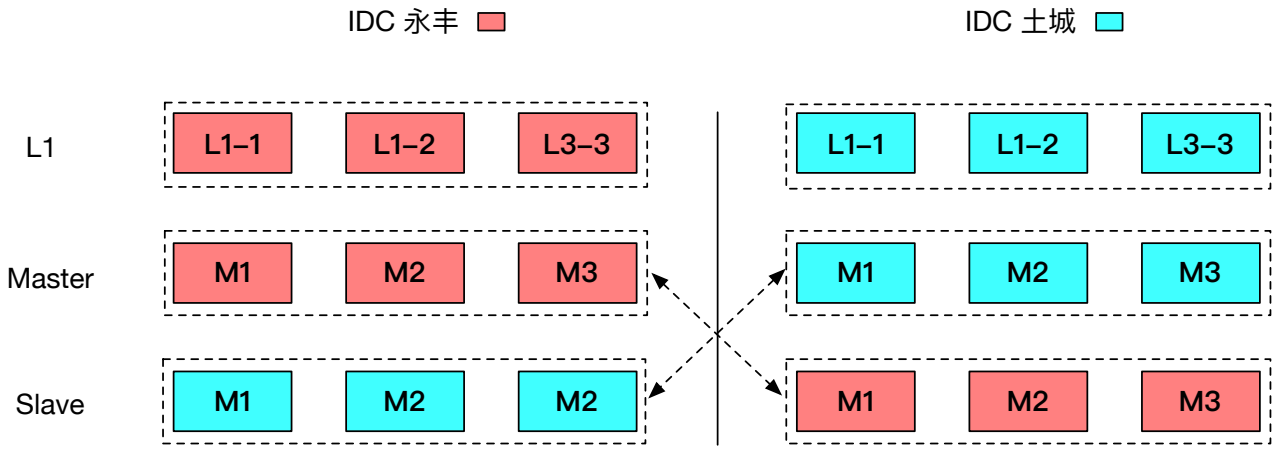


分布式缓存实现

- L1-M-S结构副本变冷问题
  - master、slave数据变冷
    - L1命中率很高，master、slave很少被访问
- master、slave做L1的逻辑分组



- slave成本问题
  - slave仅做master的备份，成本开销太大
- 多机房资源互备



分布式缓存实现

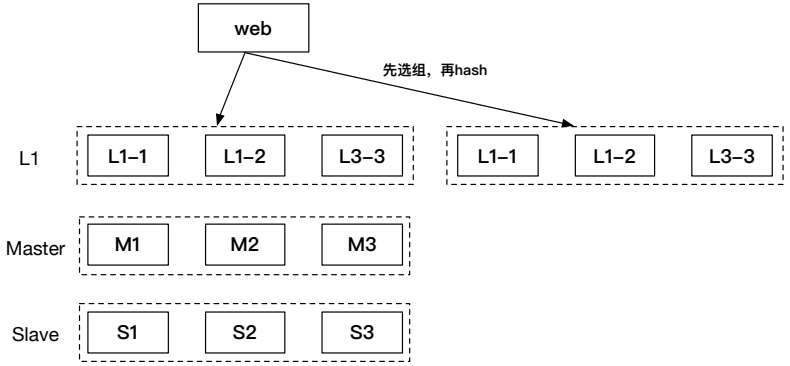
- 实现分布式的方式
  - 客户端实现
  - 服务端实现
    - redis-cluster
    - dynamo
  - 代理层实现
    - twemproxy
    - codis
    - cacheservice
  - 服务化实现
    - cacheservice

缓存服务化

- 为什么需要服务化
  - 资源配置及变更复杂
    - 资源变更：修改本地配置 -> 配置变更 -> 业务重启

```
status.dlVector.mc.master=10.211.25.236:10001,10.211.25.237:10001,10.211.25.238:10001
status.dlVector.mc.masterL1=10.211.25.209:10001,10.211.25.210:10001,10.211.16.84:10001
status.dlVector.mc.slave=10.211.25.209:10001,10.211.25.210:10001,10.211.16.84:10001
status.dlVector.mc.slaveL1=10.211.23.209:10001,10.211.23.210:10001,10.211.23.84:10001
```

  - 业务方自己实现缓存高可用逻辑，复杂度高，不易推广
    - 多级cache访问、集群同步



- 为什么需要服务化
  - 可运维性不足
    - 面向实例管理，缺少业务维度资源使用情况的掌控
    - 资源静态分配，利用率低

缓存服务化

- 缓存服务化实现
  - 配置服务化
    - 配置中心(vintage)：统一管理配置
    - 资源在线管理、动态变更配置

缓存服务化

新浪微博 weibo.com

- 缓存服务化实现
  - 访问proxy化
    - 屏蔽cache资源细节，单行配置访问
    - proxy节点无状态：线性扩展
    - 内嵌多级cache访问策略
    - 集群同步、快速预热
    - LRU: 极端热点请求

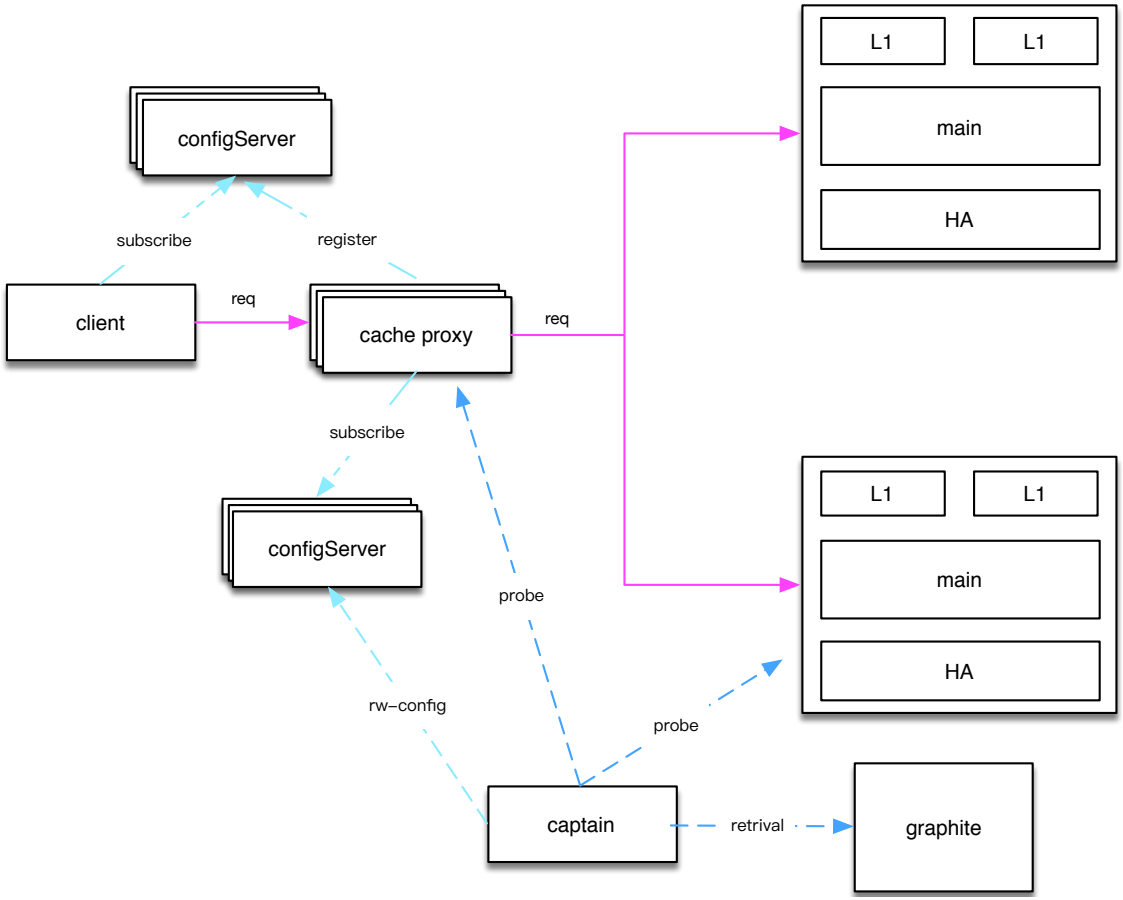
- 缓存服务化实现
  - 运维服务化(captain)
    - 统一管理缓存整个生命周期：分配、变更、回收
    - 业务维度SLA保证
    - 一键部署&升级、资源动态调整(ing)

缓存服务化

- 业务使用
  - 业务标识: groupid、namespace
    - 访问：get {namespace}key\r\n
  - 支持多种访问策略
    - localhost、dns(php)
    - 服务发现(java)

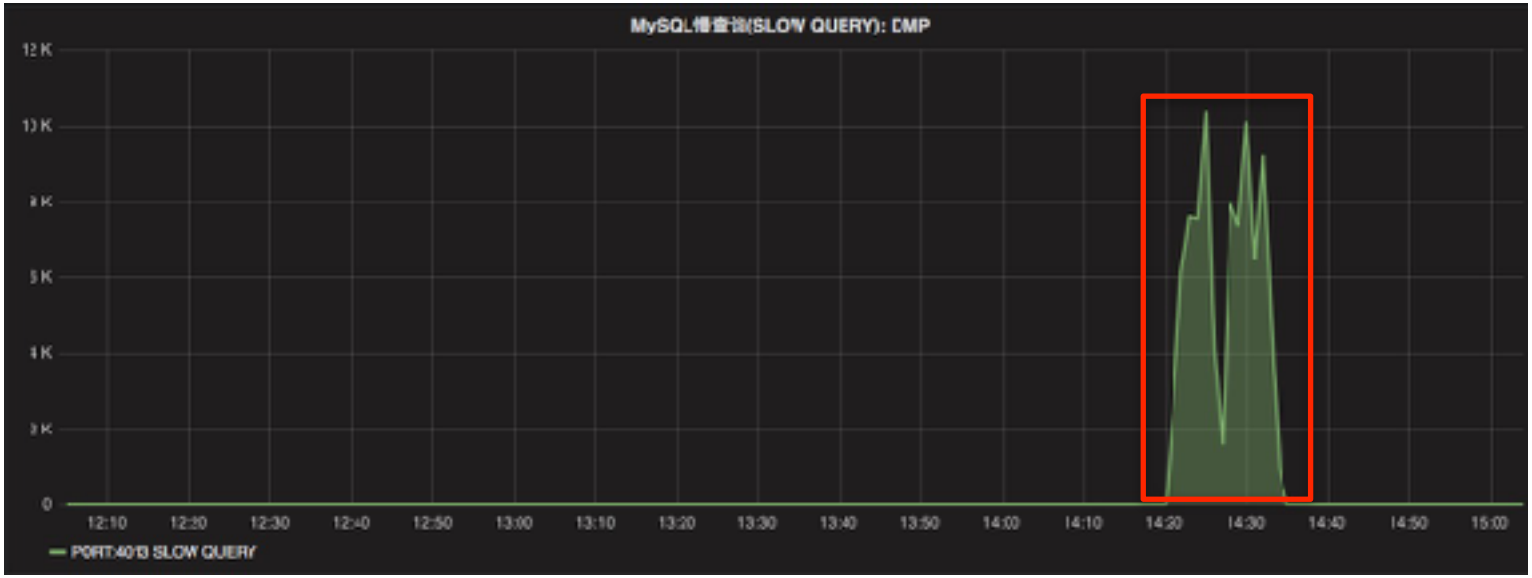
```
<weibo:cstemplate id="cacheService" namespace="user"
    expire="21600" registry="vintageTest" group="cache.service2.0.unread.pool.test"
    recovery="cacheServiceRecoveryConfigNotifer">
    <property name="useMotanMcClient" value="true" />
</weibo:cstemplate>
```

- 缓存服务化架构设计

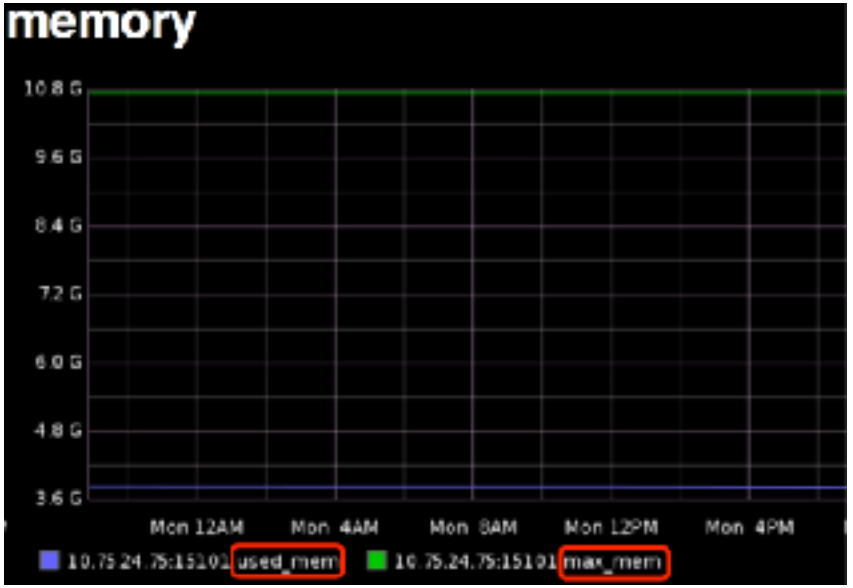

configServer
subscribe
register
client
req
cache proxy
req
L1
L1
main
HA
subscribe
configServer
probe
L1
L1
main
HA
rw-config
probe
captain
retrival
graphite

缓存实践案例

- 案例1：用户关系业务接口超时，mysql大量慢查询



缓存实践案例




```
$ perl memcached-tool 10.75.24.75:15101 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  2     120B   1296000s    3810 19480867    yes        0        0   0
  3     152B   1295998s     442 2895966     yes  6402415  1295992   0
```

```
31    80.9K        16s        1        1     yes   204910        0 1189416
```

新浪微博 weibo.com

原因分析

- lru淘汰是slabclass内部进行的
- slab分配给指定slabclass，会被该slabclass一直占用
- 业务模型变更，可用内存不足，缓存命中率下降

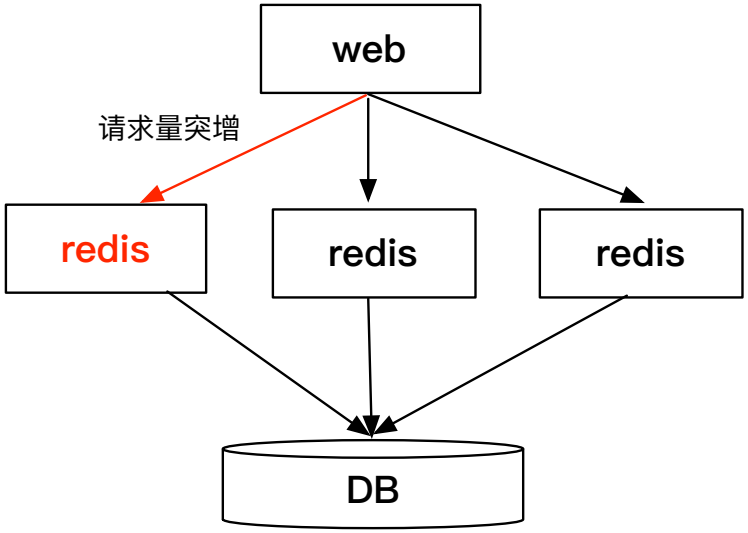
缓存实践案例

新浪微博 weibo.com

- slab calcification (slab钙化)解决方案
  - 重启
    - 简单粗暴，需要避免单点问题，避免出现雪崩
  - 随机过期
    - 过期淘汰策略也支持淘汰其他slab class的数据
    - twitter和facebook等均作了类似支持
  - 通过slab_reassign、slab_authmove
    - 1.4.11版开始支持此功能

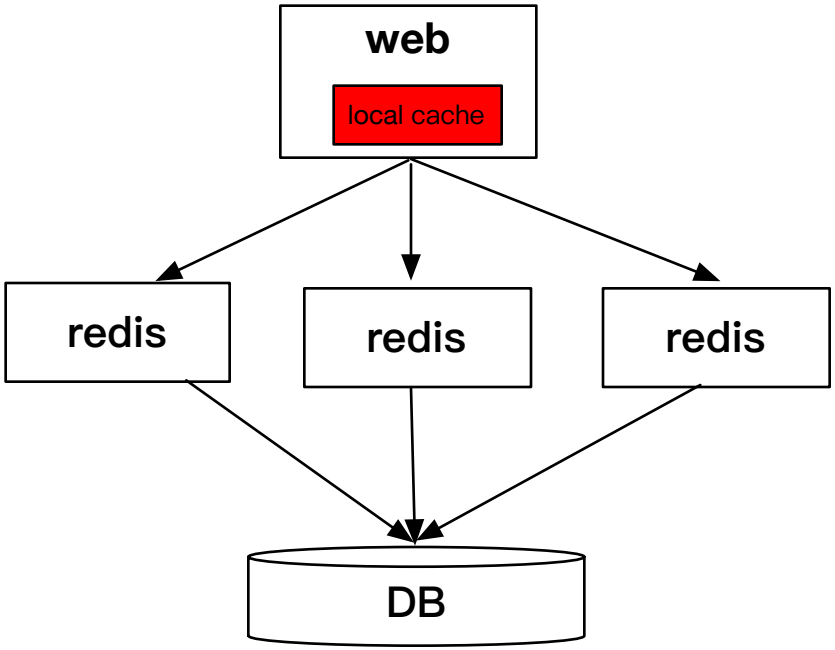
缓存实践案例

- 案例2：周一见
  - 马XX和姚XX的关注关系落在同一个redis实例
  - 突发峰值流量把redis实例打满 大量请求超时
  - 大量请求穿透至DB，DB大量超时，服务异常



新浪微博 weibo.com

- 解决方案
  - 业务层增加local cache
  - 采样获取热点数据，定时刷新



- 案例3 缓存不是万能的
  - 微博转发评论计数业务
    - 数据量巨大 千亿级
    - 每秒调用百万级
    - 响应时间 < 5ms
    - 超过一半以上的微博没有转发和评论

传统cache+DB 架构无法很好解决

新浪微博 weibo.com

- 解决方案
  - 缓存组件：cache -> storage
  - 定制组件counterservice
    - 定制存储结构，节约内存5-10倍
    - 支持冷热数据分离，ssd存放冷数据，解决容量瓶颈
    - LRU提升冷数据访问性能

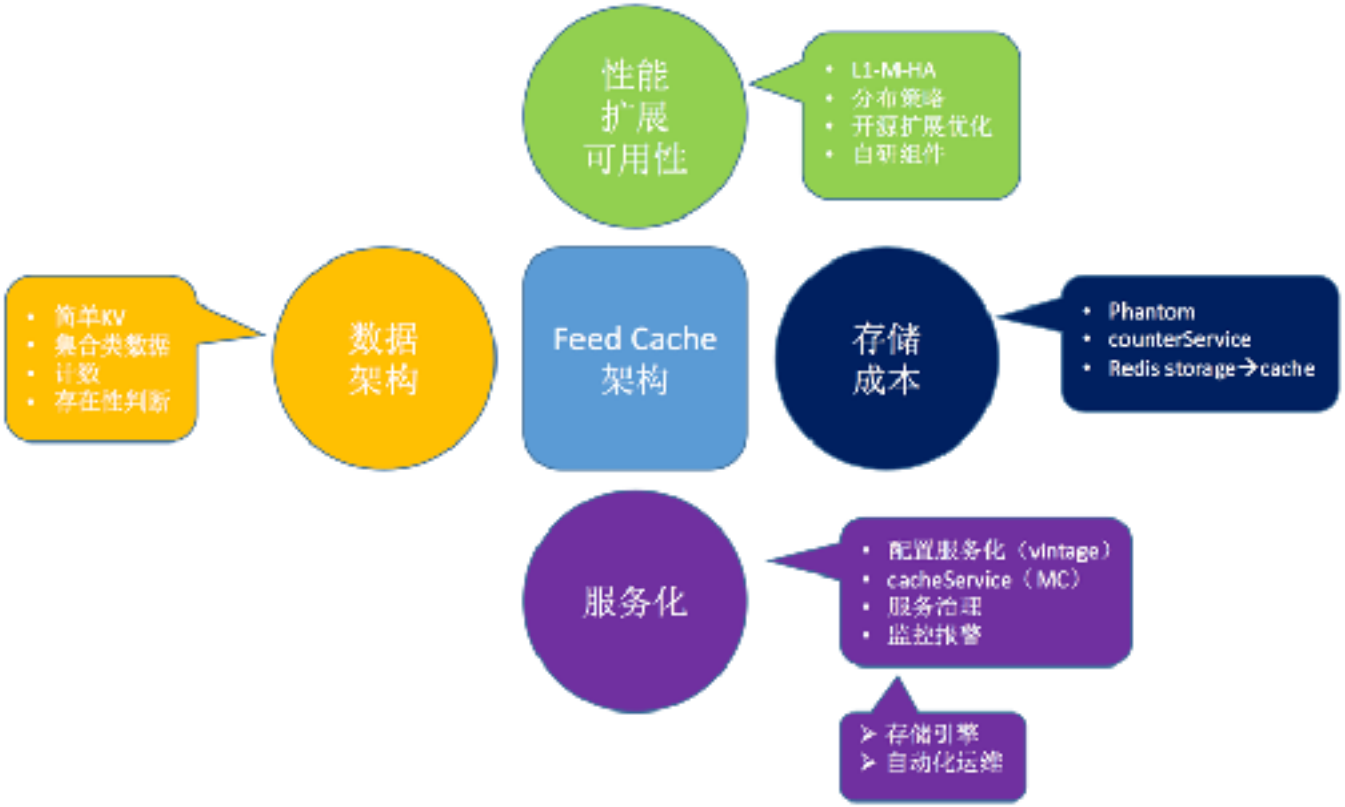
新浪微博 weibo.com

- **cache is storage**

	Memory Cache	Memory Storage
内存命中率	取决于内存容量	100%
请求访问策略	从cache获取为空穿透到db	只从Memory Storage取
过期策略	LRU	不过期
容量要求	适中?	高?(取决于空数据比例)
数据一致性	适中	强
SLA情况	适中	较好

总结

- 微博缓存架构


性能
扩展
可用性
L1-M-HA
分布策略
开源扩展优化
自研组件
简单KV
集合类数据
计数
存在性判断
数据
架构
Feed Cache
架构
存储
成本
Phantom
counterService
Redis storage→cache
服务化
配置服务化（vintage）
cacheService（MC）
服务治理
监控报警
存储引擎
自动化运维

作业

- 描述: 设计一个用户信息系统
  - 实现 增加用户信息 /users/add.json
  - 实现: 删除用户信息 /users/delete.json
  - 实现: 修改用户信息 /users/modify.json
  - 实现: 根据uid查询用户信息 /users/show.json

新浪微博 weibo.com

- 接口定义

接口名称	实现功能	请求方法	参数列表	返回值
/user/add.json	添加一个用户	POST	name：用户名 age：年龄 gender：性别	{"uid":123456,"name":"jackael","age":20,"gender":"male"}
/user/delete.json	删除一个用户	POST	uid：用户的id	{"true}
/user/update.json	更新一个用户	POST	name：用户名 age：年龄 gender：性别	{"uid":123456,"name":"jackael","age":20,"gender":"male"}
/user/show.json	获取用户信息	GET	uid：用户的id	{"uid":123456,"name":"jackael","age":20,"gender":"male"}

作业

- 要求
  - 完成用户信息系统的service层设计与开发
    - 业务逻辑（数据校验等）
    - 访问db、mc以及相关的穿透、回种逻辑的实现
  - 完成用户信息系统的缓存层设计与开发
  - 通过原始servlet提供访问接口

http://git.intra.weibo.com/bootcamp2017/distributed-cache.git

Q&A

以微博之力 让世界更美！

weibo.com