法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

分布式系统的高可用与高并发处理

墨菲定律

- 任何事情没有看起来那么简单
- 所有的事情都会比你预计的时间长
- 可能出错的事情会出错
- 如果你担心某种情况会发生，那么他一般会发生

应对高并发的基本思路

- 加快单机的速度，例如使用 Redis，提高数据访问速率；
  增加CPU的内核数，增大内存
- 增加服务器的数量，利用集群

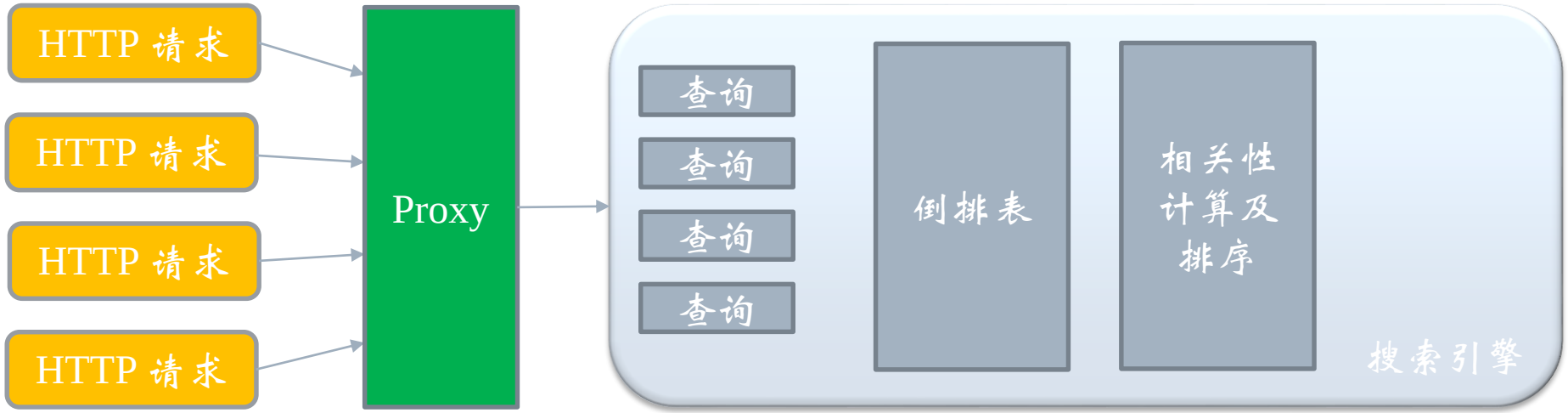
无状态

对于分布式系统的设计，应用本身没有状态，状态全部通过配置文件或者集群的服务端提供并与之同步。比如不同的机房需要读取不同的数据源，那么他们直接通过配置文件或者中心来指定

进一步，在分布式集群中，如果数据请求的节点可以做到没有状态，意味着任意节点都可以被请求，结合去中心化，就能有效避免单点的访问瓶颈

拆分

系统设计初期，不要按功能模块来进行拆分，只需要尽快保证每个开发的单元能独立运行，即按照多进程模式可以运行。例如搜索引擎的业务会分为地理信息、类别、文本等多个搜索，在初期所有搜索可以继承开发在一个HTTP请求的相应处理模块里，初期就按照业务进行拆分，会带来极大的工作量，各个模块能独立并行运行即可

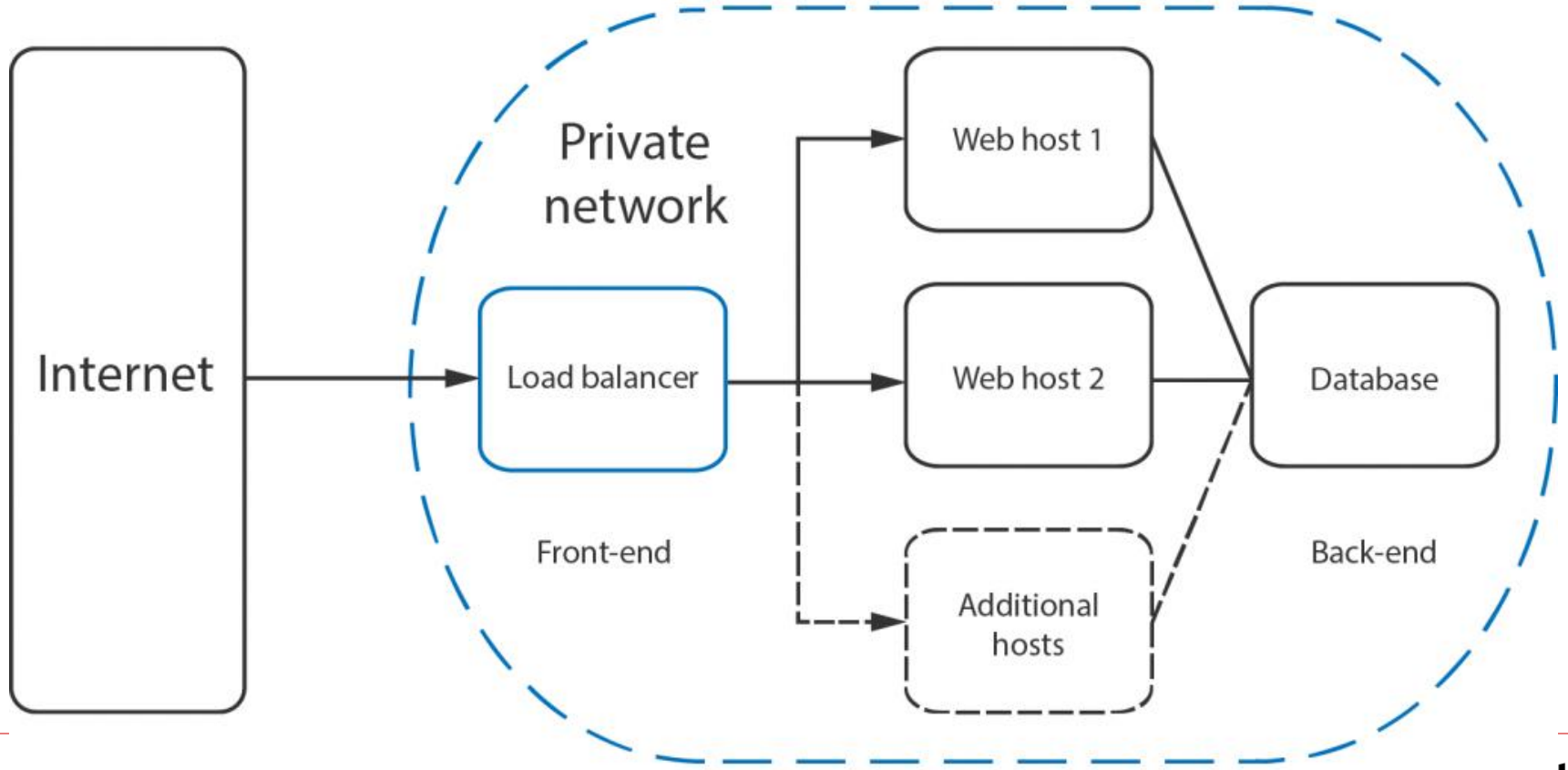


系统拆分的原则

- 功能及业务：例如搜索模块、地图模块、用户信息模块
- 子功能拆分：例如搜索模块里，图数据库与文本搜索可以分开
- 读写分离：根据读写的特性在进行拆分，例如淘宝的商品浏览与编辑，编辑是一个纯写的功能，流量是一个纯读的功能，因此可以对读写的实现进行拆分，一个集群实现商品读取的功能，而另一个规模小的多的集群提供商品写入的功能
- 代码模块：一般情况，与上图类似，一个系统可以按照 Web、Service 及 DAO 来划分，有专门负责 Web 请求的，有提供进一步数据服务的模块，以及专门的数据库HA管理模块

负载均衡

如果单机不行，需要考虑使用集群，使用集群就意味着一个负载均衡服务提供在最前端

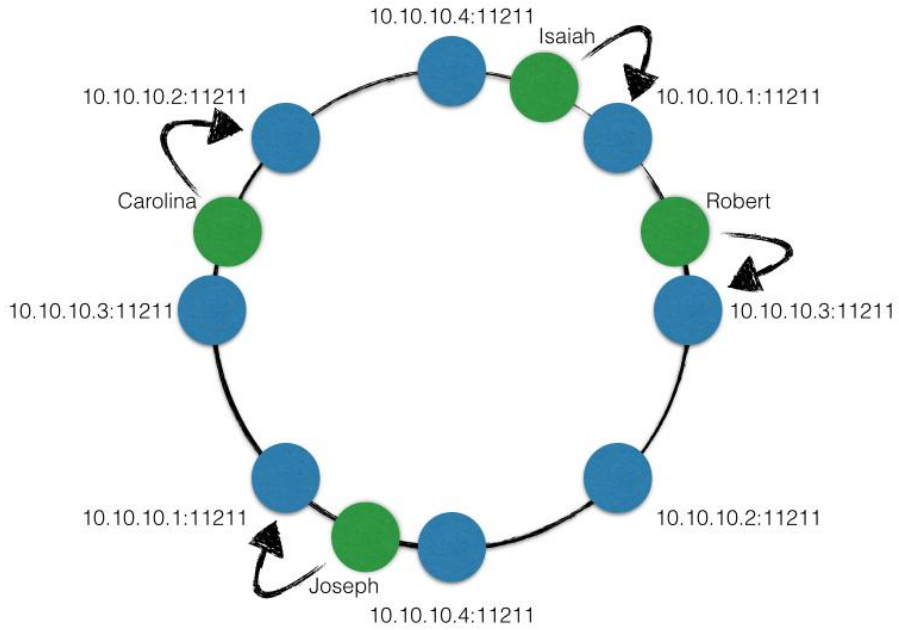


负载均衡算法

- 轮询：以轮询的方式把请求发到上游服务器，配合weight配置实现基于权重的轮询
- IP hash：根据客户的IP做负载均衡，使得胸痛Ip均衡到同一个upstream server。
- hash key：对客户端的一个key进行Hash，建议使用一致性Hash实现负载均衡。普通Hash算法，当添加或删除服务器的时候，很多key都会被重新分配到不同的服务器，使用一致性hash，使得只有极少数的key会被重新分配服务器
- least connection：把最新的请求负载均衡到活跃连接最少的服务器，依然可以与权重相结合

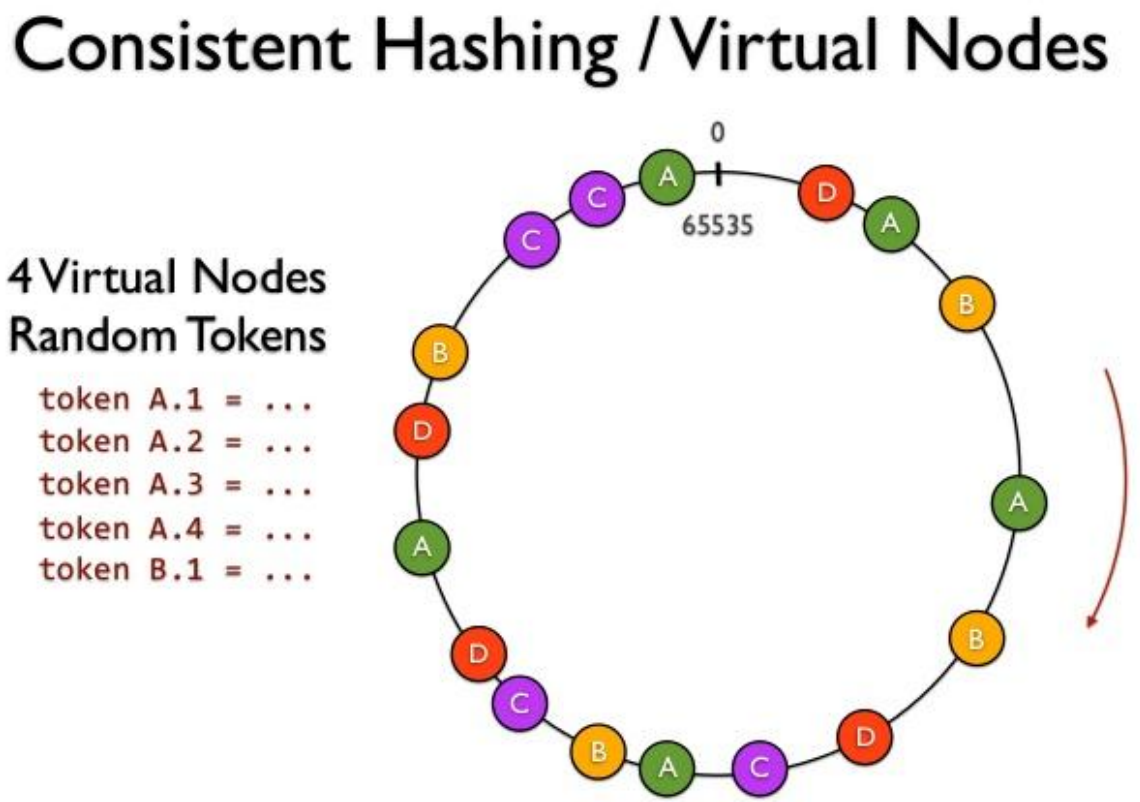
一致性Hash

一致哈希将每个对象映射到圆环边上的一个点，系统再将可用的节点机器映射到圆环的不同位置。查找某个对象对应的机器时，需要用一致哈希算法计算得到对象对应圆环边上位置，沿着圆环边上查找直到遇到某个节点机器，这台机器即为对象应该保存的位置。 当删除一台节点机器时，这台机器上保存的所有对象都要移动到下一台机器。添加一台机器到圆环边上某个点时，这个点的下一台机器需要将这个节点前对应的对象移动到新机器上。 更改对象在节点机器上的分布可以通过调整节点机器的位置来实现。



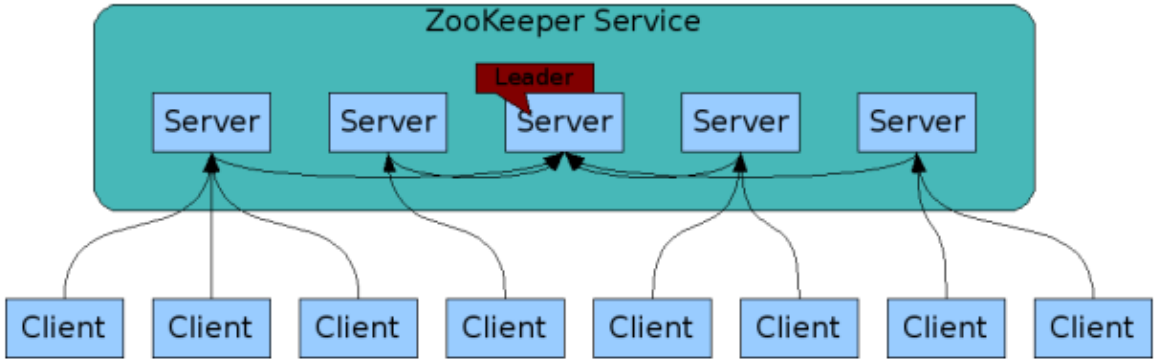
一致性Hash的平衡性

通过增加虚拟节点，使得节点的分布及Hash算法能实现平衡



服务化

当服务集群设计为主从架构的时候，意味着需要有多个主服务器提供备份或容灾，这样就要考虑使用服务的自动注册和发现，比如使用 ZooKeeper



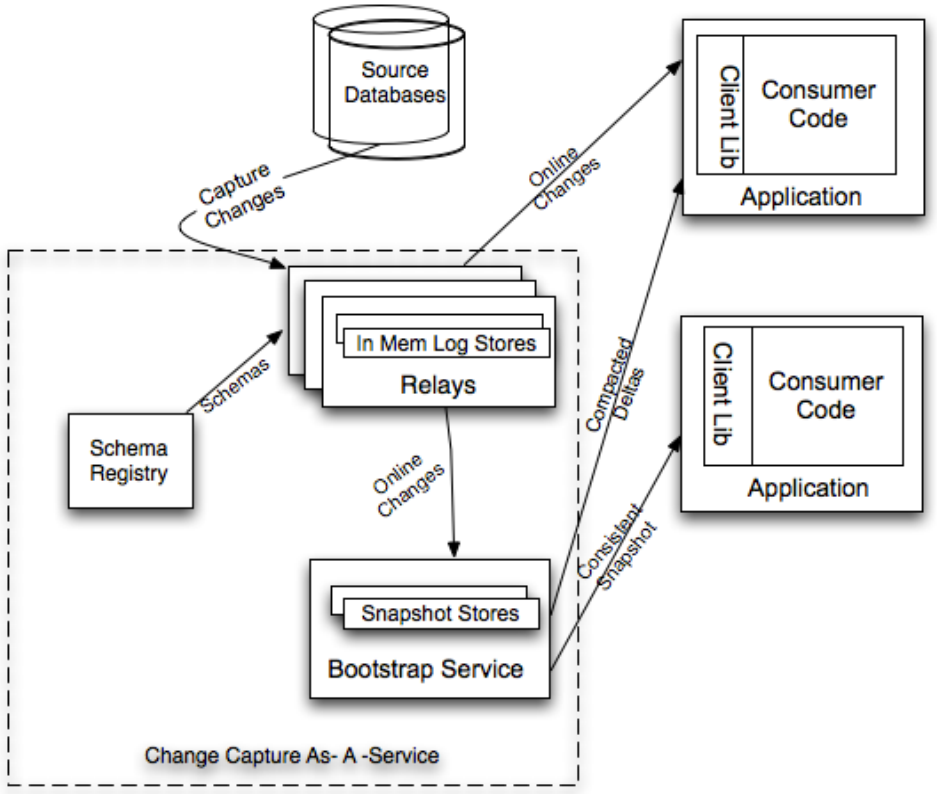
消息队列

消息队列用来解耦一些不需要同步调用的服务或者订阅一些通知以及系统的变化。使用消息队列可以实现服务解耦、异步处理、流量缓冲等。案例：

- 日志系统：日志系统一般是同步的，本身会有一个缓冲区，缓冲区满的时候再写入到磁盘。高峰期，log系统写磁盘的时候会造成线程的阻塞，可以通过消息发送给logger，由logger来处理
- 抢购：抢购的流量会突然增加，所以把所有抢购的请求排队，防止数据库的访问请求过载
- 数据同步：降MySQL的数据变更同步到 Redis 里，可以使用消息队列，根据日志系统记录的顺序，依次对数据进行同步操作
- 任务队列：新用户注册后，一般会发送短信、邮件或者标示特权等，此时可以用任务队列来完成这些工作，而不是在主线程里全部处理完才返回给用户
- 订阅系统：当系统状态发生变化的时候，需要消息同步的模块可以通过消息订阅的方式来取得这些更新

databus 架构

我们可以通过 databus 来实现 MySQL 数据与 Redis 的更新操作，把 MySQL 的数据变化同步到 Redis



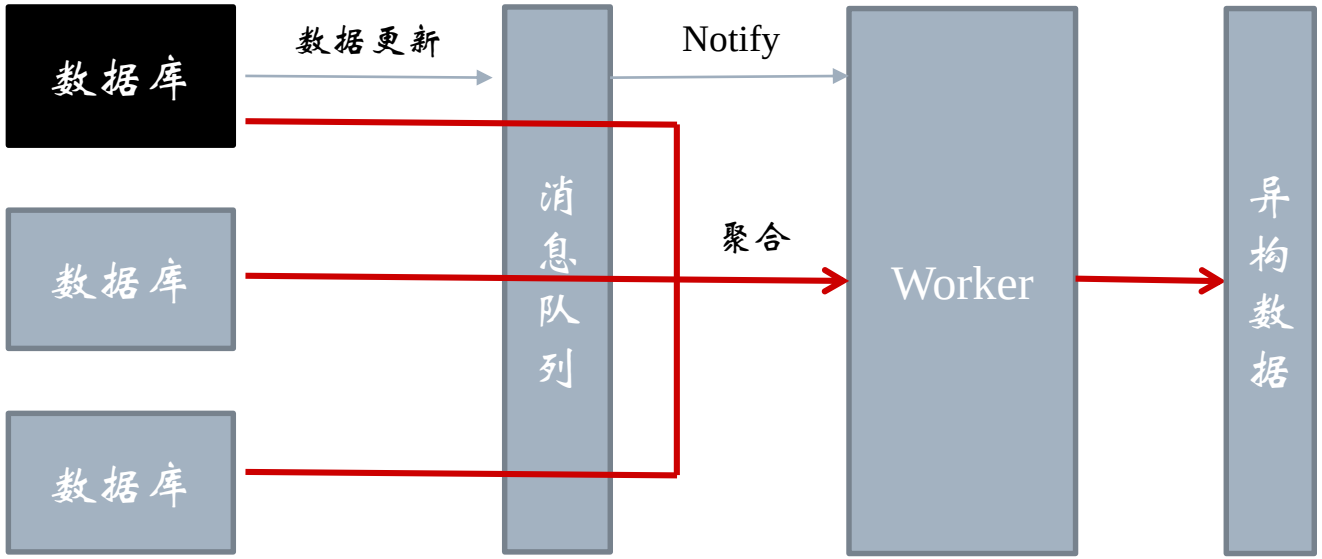
数据异构

数据异构，是指把一些联合查询的数据，直接组合成一张结果表，这样在读取的时候能更快速取回结果。通常这样的表会比较大，因此需要按照主键做分库分表来增加读取的性能，带来的问题：

- 数据同步
- 某一项服务的终止导致整体服务终止
- 聚合查询

数据异构实现

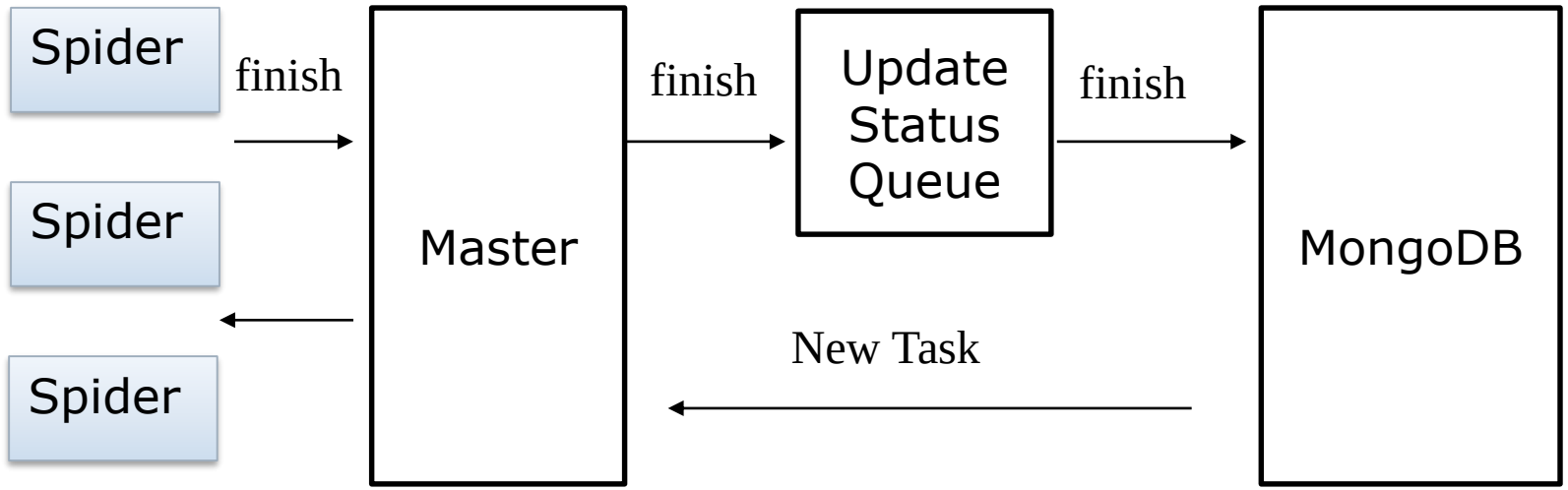
1. 通过 MQ 机制接收数据变更，然后原子化存储
2. 在任务队列里，聚合数据源并更新
3. 聚合的数据，根据类型、实例进行分割，分割的数据还可以考虑分片



案例：爬虫的串行化处理

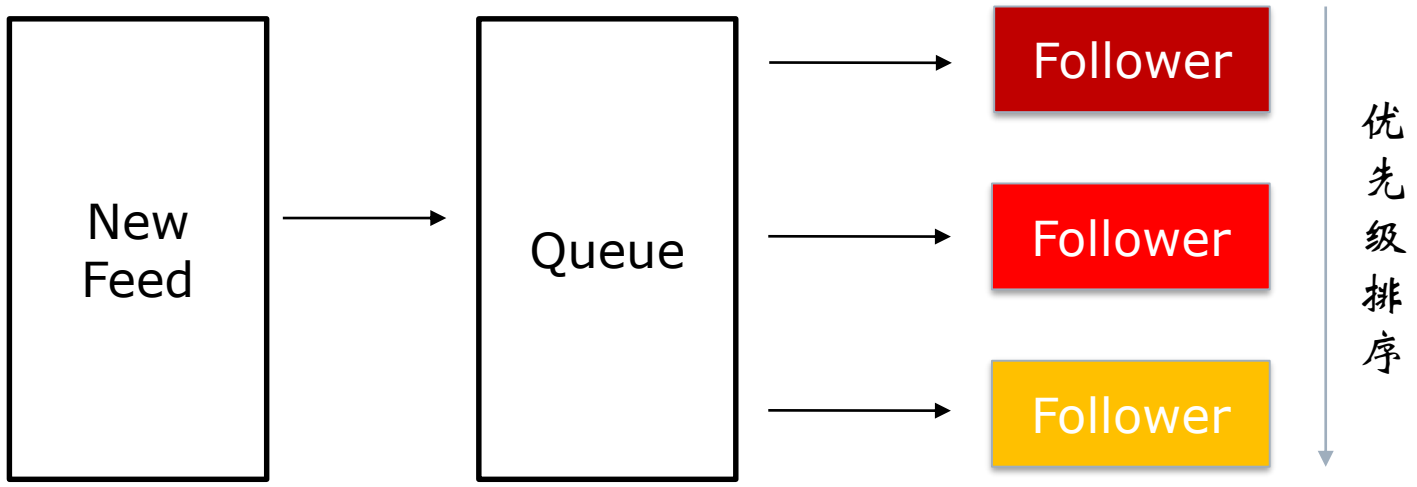
为了防止数据库被过度访问，可以由前端统一管理任务队列，这时，可以在前端消息请求与数据库之间再建立一层任务队列，用于隔离一部分的数据更新操作

- 将强一致性比较弱的更新操作，通过队列来异步完成
- 提高需要立即返回的获取任务请求的线程优先级，使得这个读写操作能优先处理



案例：微博的Feed流推送

- 50,000,000用户使用新浪微博
- 最高发表3000条微博/秒
- 姚晨发表一条微博，会被6,689,713粉丝读到(11/10/2010)

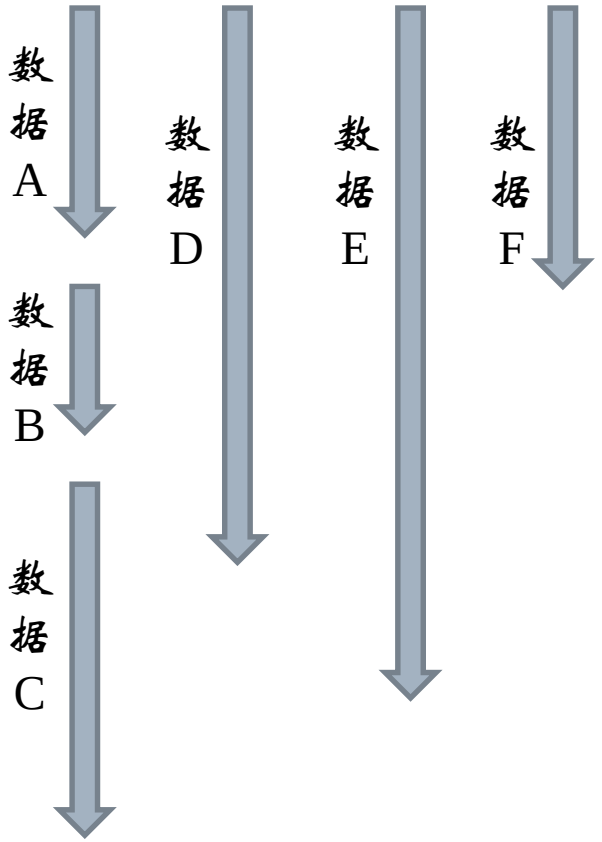


缓存

缓存不仅仅存在于服务端，它可以被用在从最前端到最后端的任意节点，只要缓存被命中，对应的数据查询就可以得到量级的提升

- 客户端缓存：浏览器的cookie、storage，APP 的本地数据存储
- 网络层：CDN、镜像服务器、P2P 技术等
- 服务前端：接入层的缓存，应用层缓存
- 分布式缓存：利用 Redis 数据库实现分布式的缓存系统
- 数据库缓存：当所有结果都没命中的情况下，可以依赖数据库的缓存提高检索排序效率

并发



假设一个请求，需要同时获得数据A、B、C、D、E和F，其中B的数据需要依赖数据A，数据C的产生需要依赖B，可以按照左边图的方式来做处理

隔离

当海量服务请求发送到前端的时候，单机的故障率会极大提高，此时一旦一台服务器出故障停止响应，前端反向代理会把请求发送到集群的其它节点，导致整体负载的进一步提升，使得所有的服务请求都被拖累，使得服务器节点如雪崩一样纷纷过载从而导致服务不可用

因此需要考虑从以下几个维度对访问进行隔离：

1. 资源的隔离：比如 js css 这类访问量极大的文件放到CDN上
2. 热点的隔离：一些高频访问的应用，需要的数据库部署在单独的集群下
3. 读写分离：把读的数据放到Redis集群，通过databus把数据库的数据同步到Redis，Redis 本身是主从模式的，这样海量的读操作可以得到最快的响应

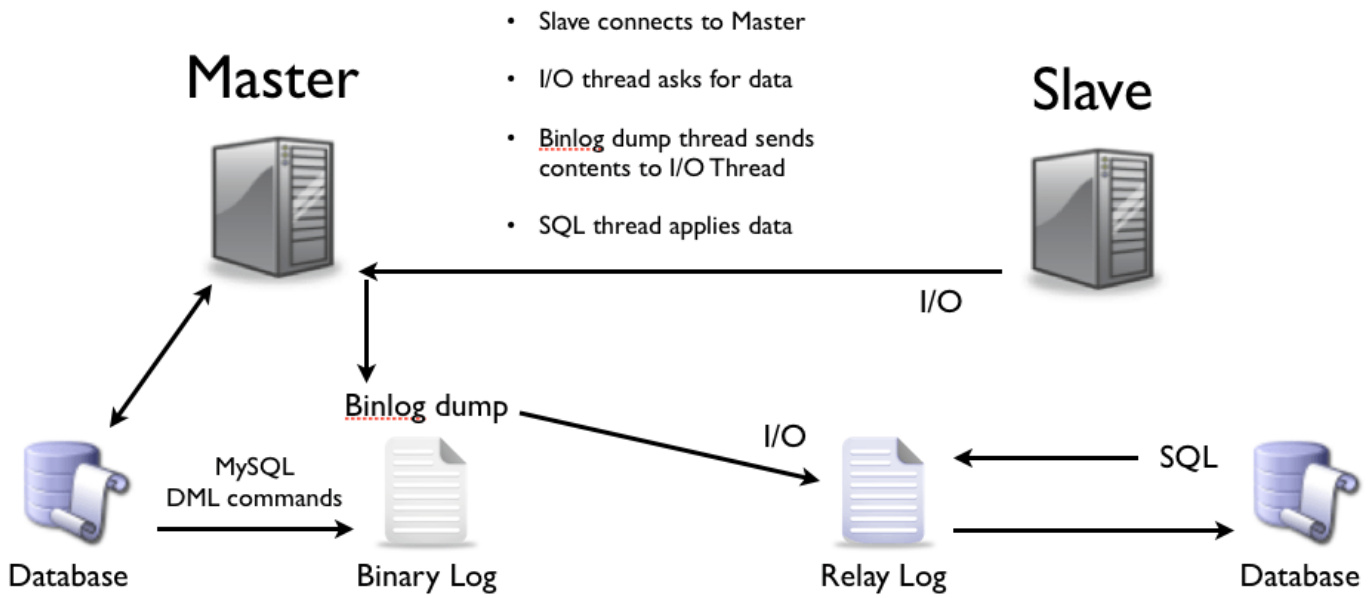
读写分离 I – 性能优势分析

- 物理服务器增加，负荷增加
- 主从只负责各自的写和读，极大程度的缓解X锁和S锁争用
- 从库可配置 myisam 引擎，提升查询性能以及节约系统开销
- 从库同步主库的数据和主库直接写还是有区别的，最重要区别在于主库向从库发送binlog是异步的，从库恢复数据也是异步的
- 读写分离适用与读远大于写的场景，如果只有一台服务器，当select很多时，update和delete会被这些select访问中的数据堵塞，等待select结束，并发性能不高。 对于写和读比例相近的应用，应该部署双主相互复制
- 可以在从库启动是增加一些参数来提高其读的性能，例如--skip-innodb、--skip-bdb、--low-priority-updates以及--delay-key-write=ALL。当然这些设置也是需要根据具体业务需求来定得，不一定能用上MySQL复制另外一大功能是增加冗余，提高可用性，当一台数据库服务器宕机后能通过调整另外一台从库来以最快的速度恢复服务，因此不能光看性能，也就是说1主1从也是可以的。

读写分离 II - 主从同步

读写分离是通过主从服务器同步来实现的，主服务器负责写任务，而多个从服务器构成的集群负责读取任务。

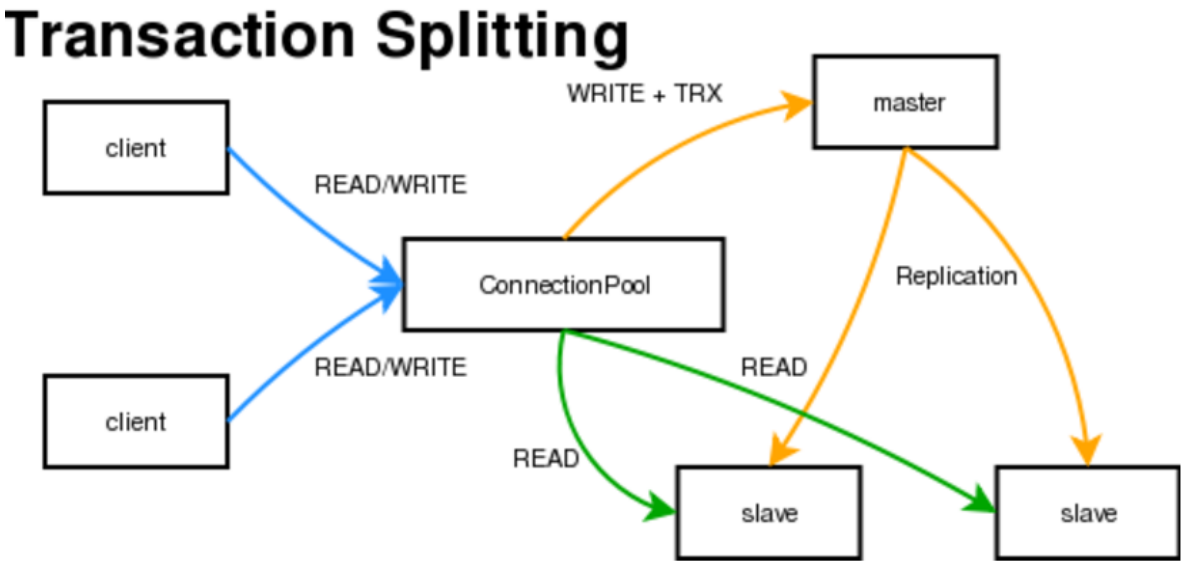
主服务器先从binlog读到变化，随后从服务器得到变化，发起请求到主服务器，获得改动的binlog位置，随后发起同步 binlog 的请求，将所有写操作同步到自己的服务器。



读写分离 III - HAProxy

- 读写分离需要处理一个比较复杂的机制，Master-Slave 异步复制存在延迟。所以，读、写分离时，一般的做法是，前端程序加判断，首先检查SLAVE节点同步位置以及状态是否同步至最新，确认其正常后，然后将查询请求发送至此节点

- 将实时性、一致性没有要求的请求直接发往从数据库，而实时性要求高的，通过 Mysql Proxy 请求或者直接向主数据库请求



分库分表

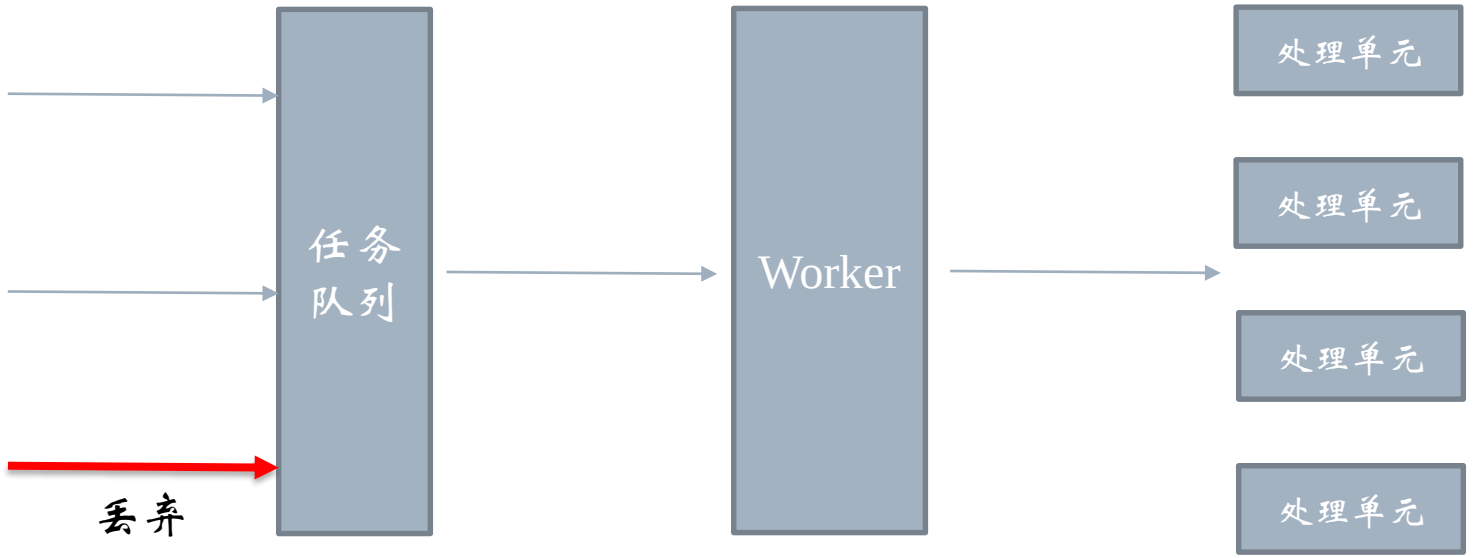
对于NoSQL的数据库，往往天然支持分布式部署，例如 HBASE、MongoDB，因此直接可以用集群，通过分片及复制来提高响应能力

对于RDBMS，不能进行分片，所以当容量增长的时候，需要考虑分表，把旧的内容独立出来，新建一张表存储；当业务增长很快，可以进一步考虑把旧的历史数据存储到单独的服务器上的一个数据库，由应用层根据请求来判断到哪个库上读取数据，这样能使得热点数据的总量以及总访问量是一定的

分库分表的策略

- 取模：按照主键取模后做分库分表，缺点是按照非主键查询的时候需要跨库跨表的查询，扩容需要建立新的集群并进行数据迁移
- 分区：例如2B的系统，每个大的客户一张表，每个客户下的分店可以考虑分表；根据时间或地区，进行表或数据库的切割

限流



HTTP的请求全部发送到任务队列中，当队列满的时候，直接返回服务超时；队列的大小按照并发处理能力来设置，使得进入队列的请求都能在一定时间内响应

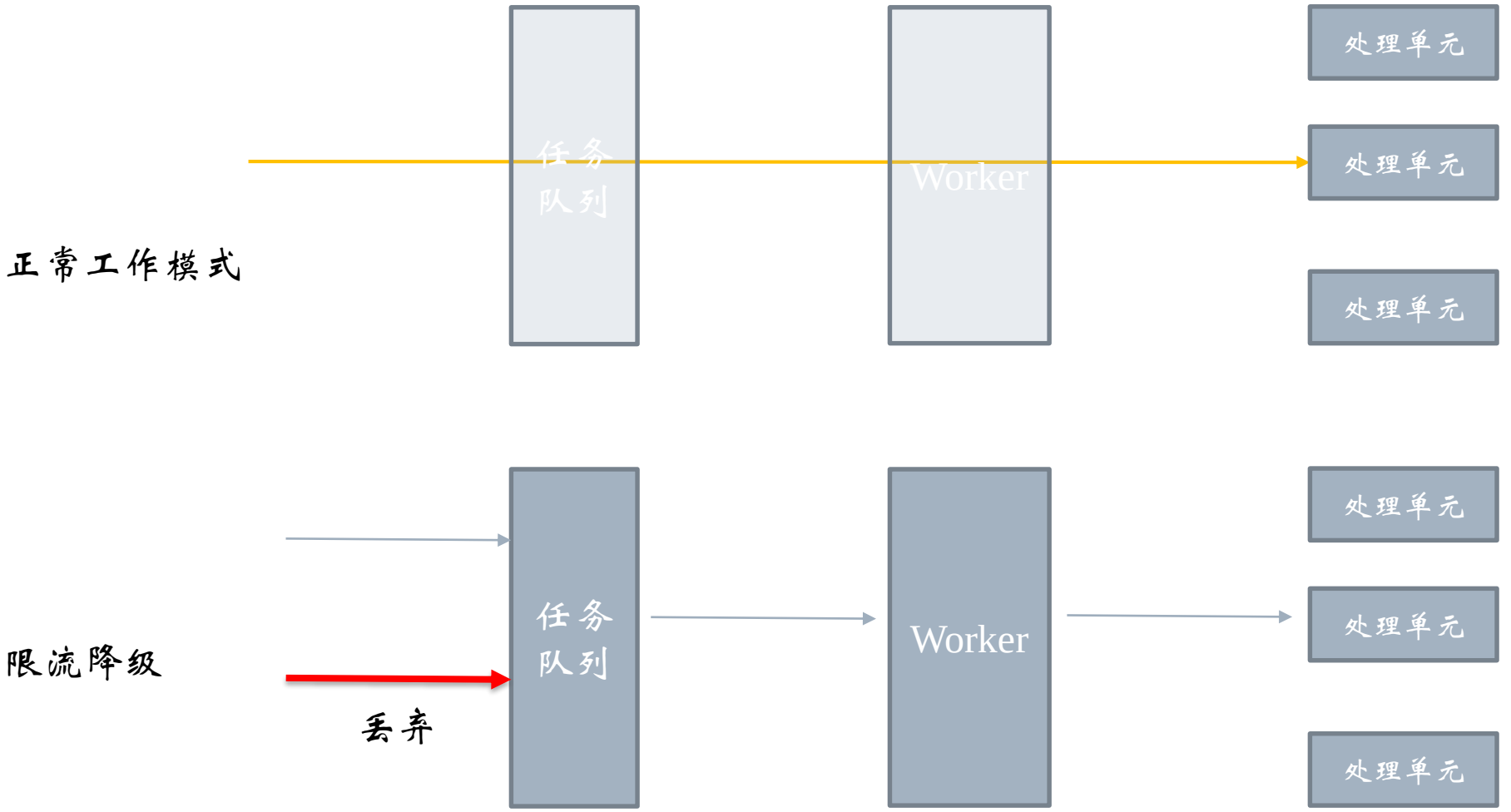
降级

- 降级是指的在服务并发突然增大的时候，启动限制性措施，降低服务提供的能力从而保护系统性能。降级一般是在系统的接口之间，设置降级开关，在特定的时候启用。例如前面讲到为限流而做的缓冲任务队列，可以通过降级来启用或停用：平峰和低谷期请求通过代理层直达后端；高峰期通过降级，请求都进入到队列，由worker从队列获取数据来请求。
- 降级措施还包括针对服务级别的，例如在高峰情况下只读缓存，缓存失效的情况下直接返回失败，使得热点的访问能被响应，而非热点数据请求被拒绝；微博打开个人主页的时候，还会出现热点推荐、广告等，如果出现流量暴涨，可以关闭这些功能的服务接口

降级的启动方式

- 超时：数据库、HTTP服务或者远程调用超时后，可以自动触发降级。例如淘宝的商品主页会异步加载价格、评论、推荐，而抢购活动时，评论和推荐都可以临时不展示，通过超时来自动降级
- 统计失败次数：调用外部服务，例如数据来源于图数据库的情况，失败次数达到一定次数的时候自动降级
- 故障：远程服务出错，可以直接降级并报警
- 限流：流量监控，超过峰值后自动开启降级，把所有请求排队

自动化限流降级


正常工作模式
任务队列
Worker
处理单元
处理单元
处理单元
限流降级
丢弃
任务队列
Worker
处理单元
处理单元
处理单元

Thanks!

Q&A