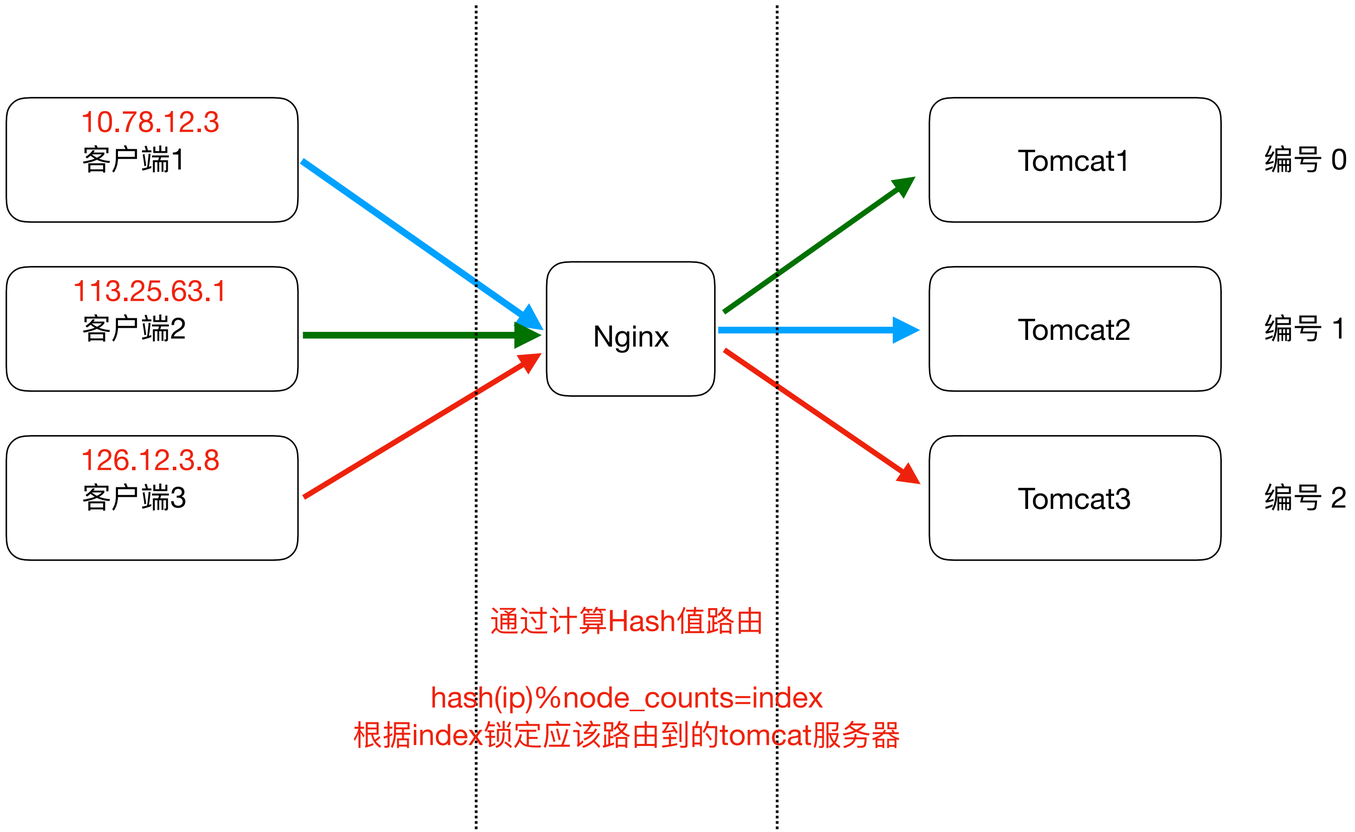
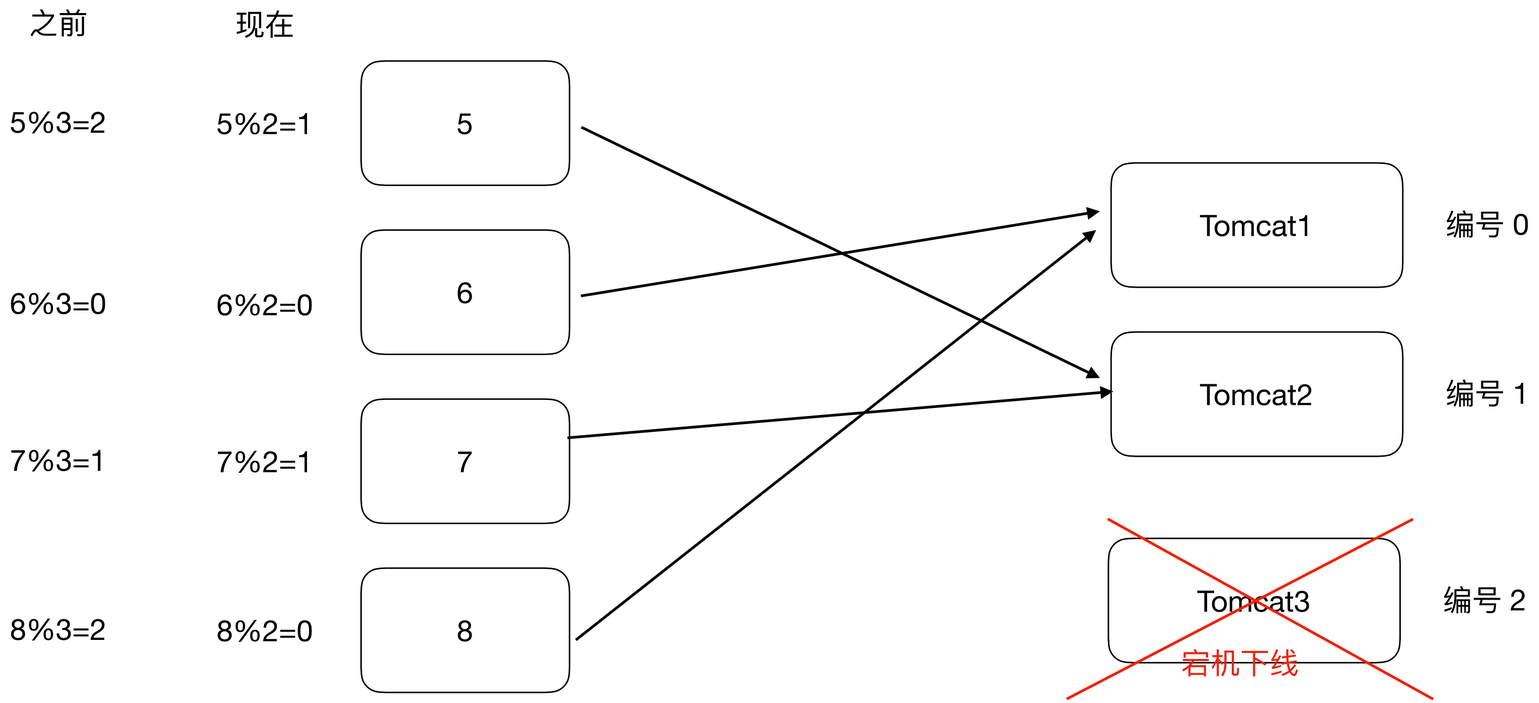
by 应癫

Nginx ip_hash策略示意图

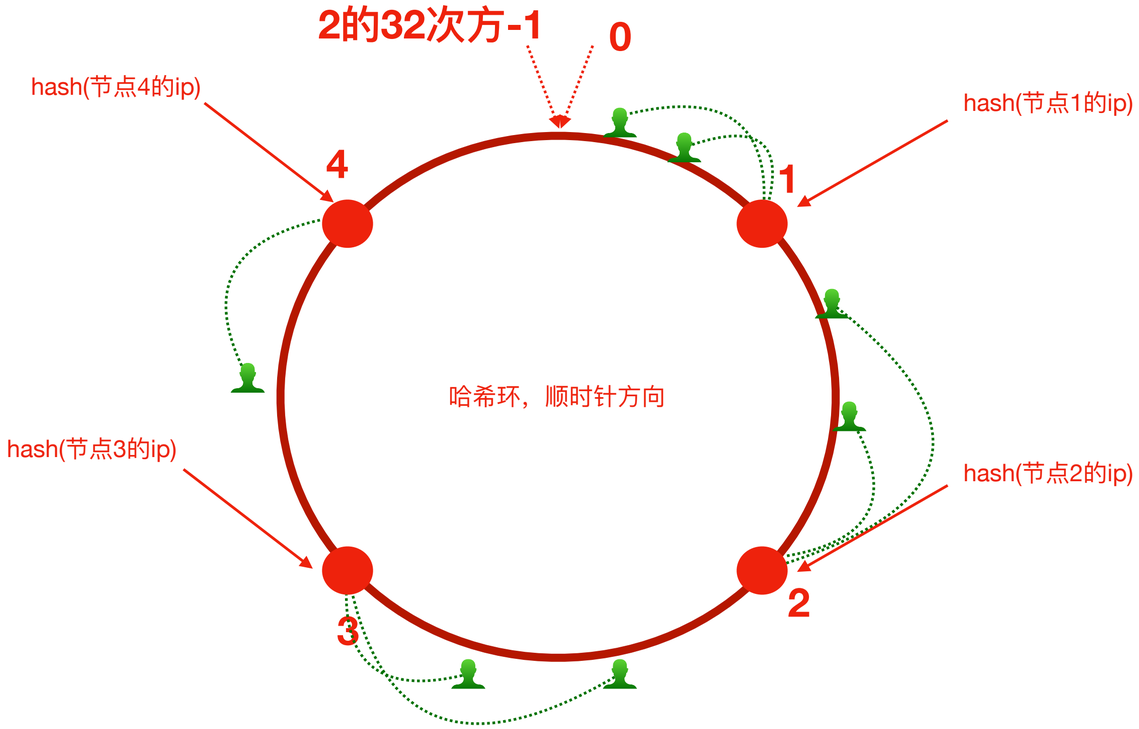


by 应癫　　普通Hash算法存在的问题示意图

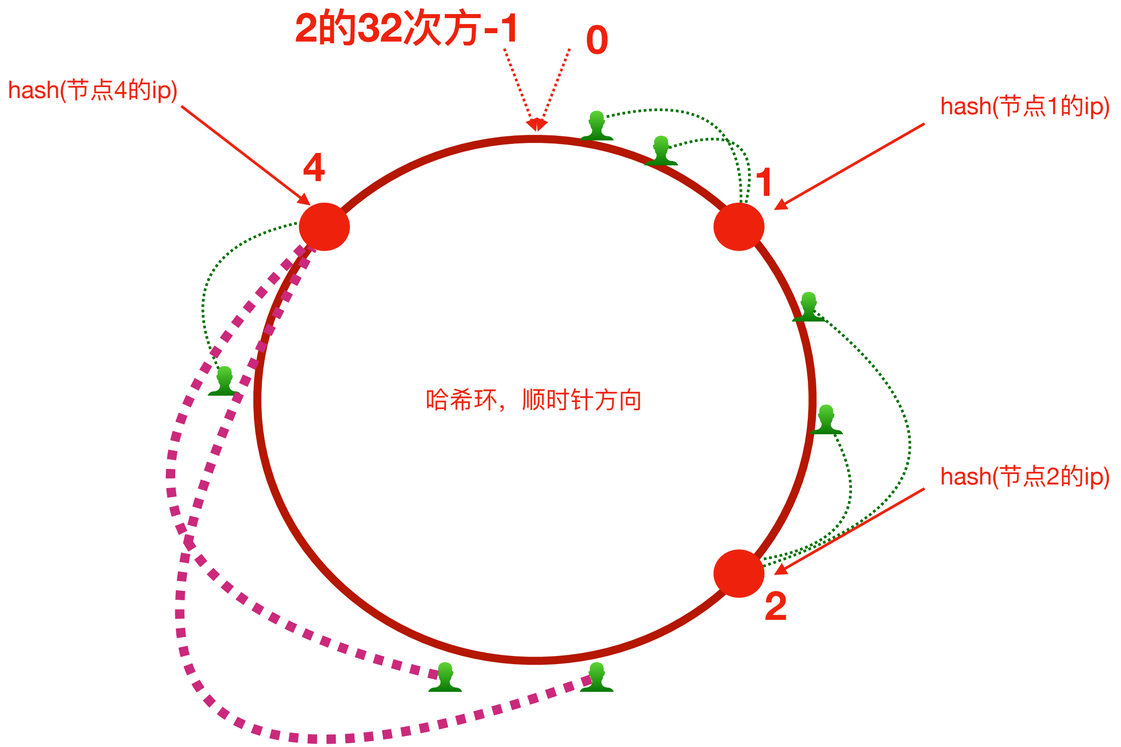


by 应癫

一致性hash算法示意

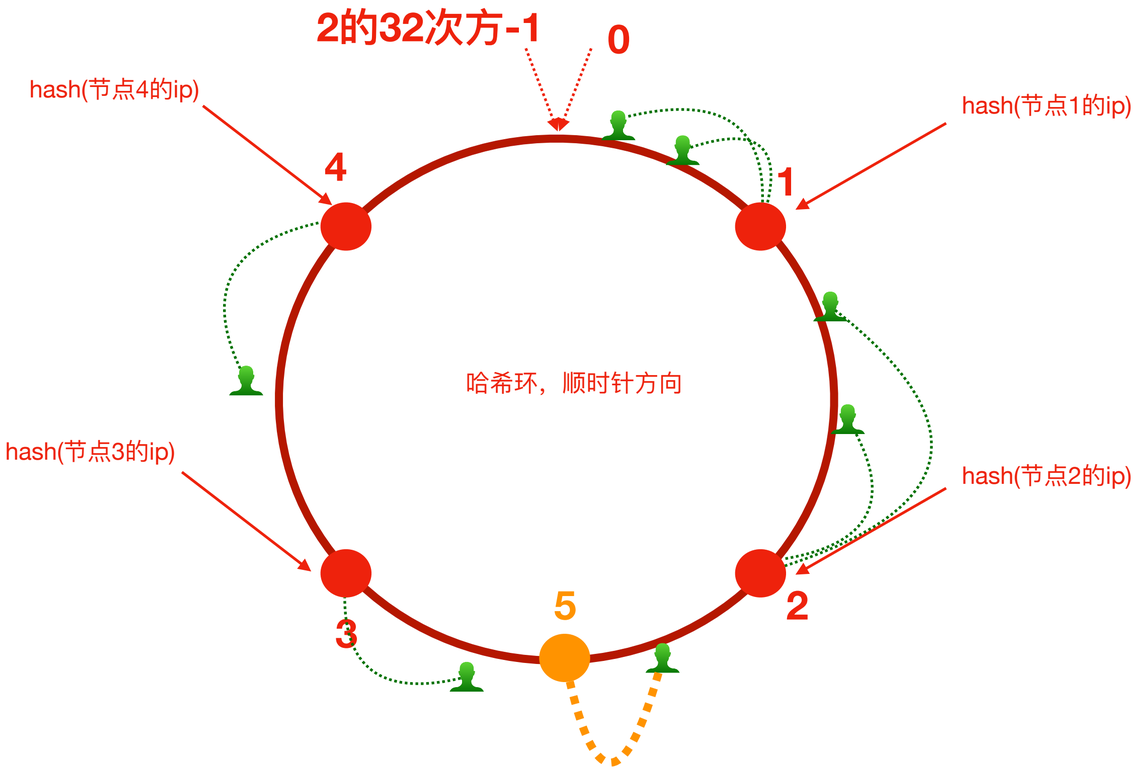


by 应癫　　一致性hash算法（缩容－减少服务器）示意



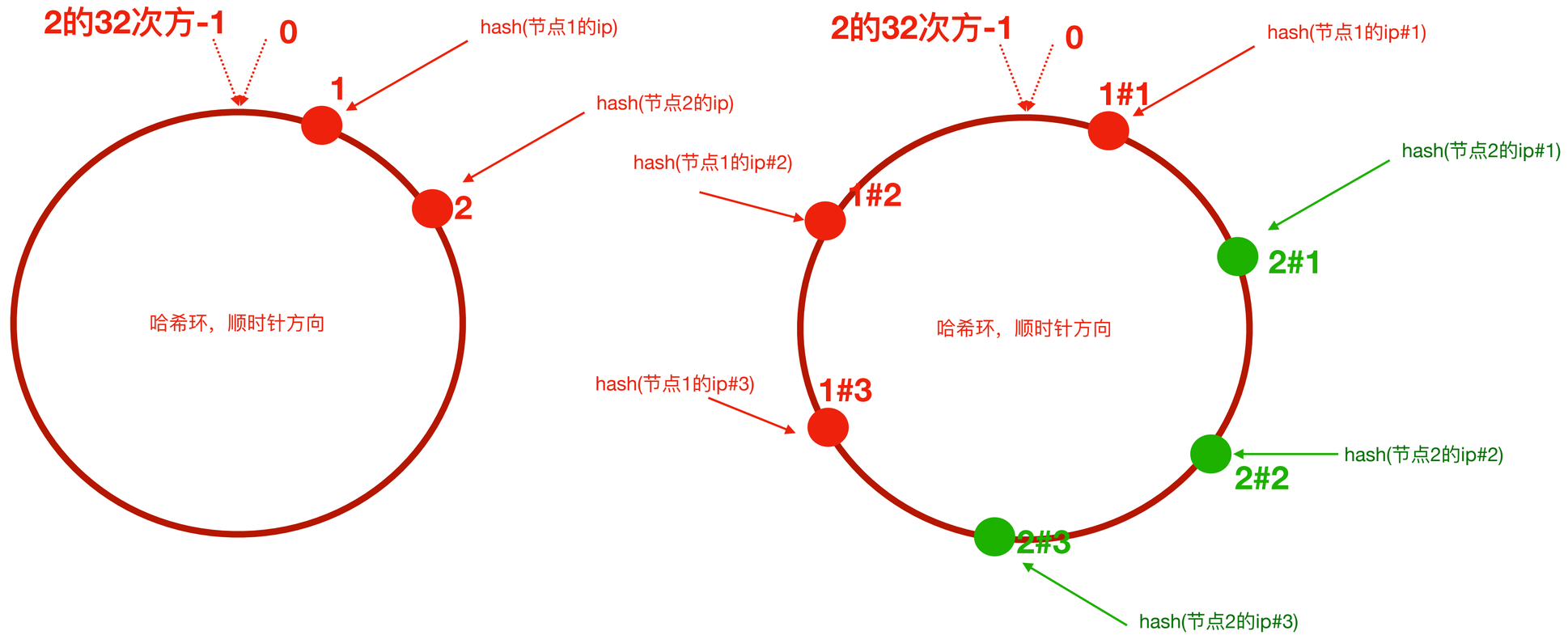
服务器3下线后，原来路由到3的客户端
重新路由到服务器4，对于其他客户端没有影响
只是这一小部分受影响（请求的迁移达到了最小，这样的算法对分布式集群来说非常合适的，避免了大量请求迁移 ）

by 应癫　　一致性hash算法（扩容—增加服务器）示意



增加服务器5之后，原来路由到3的部分客户端
路由到新增服务器5上，对于其他客户端没有影响
只是这一小部分受影响（请求的迁移达到了最小，这样的算法对分布式集群来说非常合适的，避免了大量请求迁移 ）

by 应癫　　一致性hash算法+虚拟节点方案示意



1）如前所述，每一台服务器负责一段，一致性哈希算法对于节点的增减都只需重定位环空间中的一小部分数据，具有较好的容错性和可扩展性。

但是，一致性哈希算法在服务节点太少时，容易因为节点分部不均匀而造成数据倾斜问题。例如系统中只有两台服务器，其环分布如下，节点2只能负责非常小的一段，大量的客户端请求落在了节点1上，这就是**数据倾斜问题**

2）为了解决这种数据倾斜问题，一致性哈希算法引入了虚拟节点机制，即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个此服务节点，称为虚拟节点。

具体做法可以在服务器ip或主机名的后面增加编号来实现。比如，可以为每台服务器计算三个虚拟节点，于是可以分别计算 “节点1的ip#1”、“节点1的ip#2”、“节点1的ip#3”、“节点2的ip#1”、“节点2的ip#2”、“节点2的ip#3”的哈希值，于是形成六个虚拟节点，当客户端被路由到虚拟节点的时候其实是被路由到该虚拟节点所对应的真实节点

by 应癫　　Nginx配置一致性hash负载均衡策略

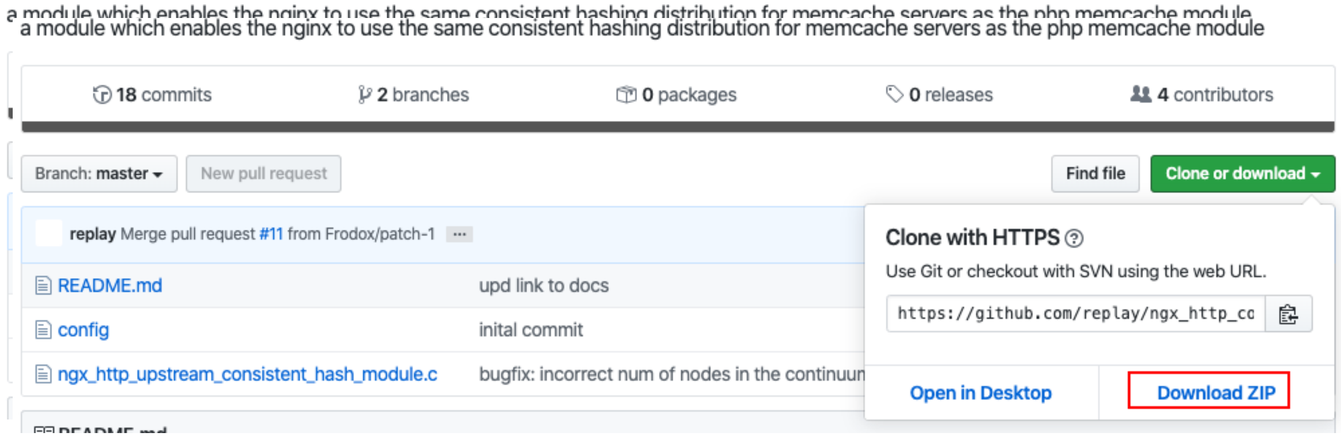
ngx_http_upstream_consistent_hash 模块是一个负载均衡器，使用一个内部一致性hash算法来选择合适的后端节点。
该模块可以根据配置参数采取不同的方式将请求均匀映射到后端机器，
　consistent_hash $remote_addr：可以根据客户端ip映射
　consistent_hash $request_uri：根据客户端请求的uri映射
　consistent_hash $args：根据客户端携带的参数进行映射
示例如下：

```
# 负载均衡的配置
upstream  lagouServer {
    consistent_hash $request_uri;
    server 127.0.0.1:8080;
    server 127.0.0.1:8082;
}
```

1）如前所述，每一台服务器负责一段，一致性哈希算法对于节点的增减都只需重定位环空间中的一小部分数据，具有较好的容错性和可扩展性。
但是，一致性哈希算法在服务节点太少时，容易因为节点分部不均匀而造成数据倾斜问题。例如系统中只有两台服务器，其环分布如下，节点2只能负责非常小的一段，大量的客户端请求落在了节点1上，这就是**数据倾斜问题**
2）为了解决这种数据倾斜问题，一致性哈希算法引入了虚拟节点机制，即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个此服务节点，称为虚拟节点。
具体做法可以在服务器ip或主机名的后面增加编号来实现。比如，可以为每台服务器计算三个虚拟节点，于是可以分别计算 “节点1的ip#1”、“节点1的ip#2”、“节点1的ip#3”、“节点2的ip#1”、“节点2的ip#2”、“节点2的ip#3”的哈希值，于是形成六个虚拟节点，当客户端被路由到虚拟节点的时候其实是被路由到该虚拟节点所对应的真实节点

by 应癫　　Nginx配置一致性hash负载均衡策略步骤

1）github下载nginx一致性hash负载均衡模块 https://github.com/replay/ngx_http_consistent_hash



2）将下载的压缩包上传到nginx服务器，并解压

3）我们已经编译安装过nginx，此时进入当时nginx的源码目录，执行如下命令

```
./configure —add-module=/root/ngx_http_consistent_hash-master
make
make install
```

4）Nginx就可以使用啦，在nginx.conf文件中配置

```
# 负载均衡的配置
upstream  lagouServer {
    consistent_hash $request_uri;
    server 127.0.0.1:8080;
    server 127.0.0.1:8082;
}
```

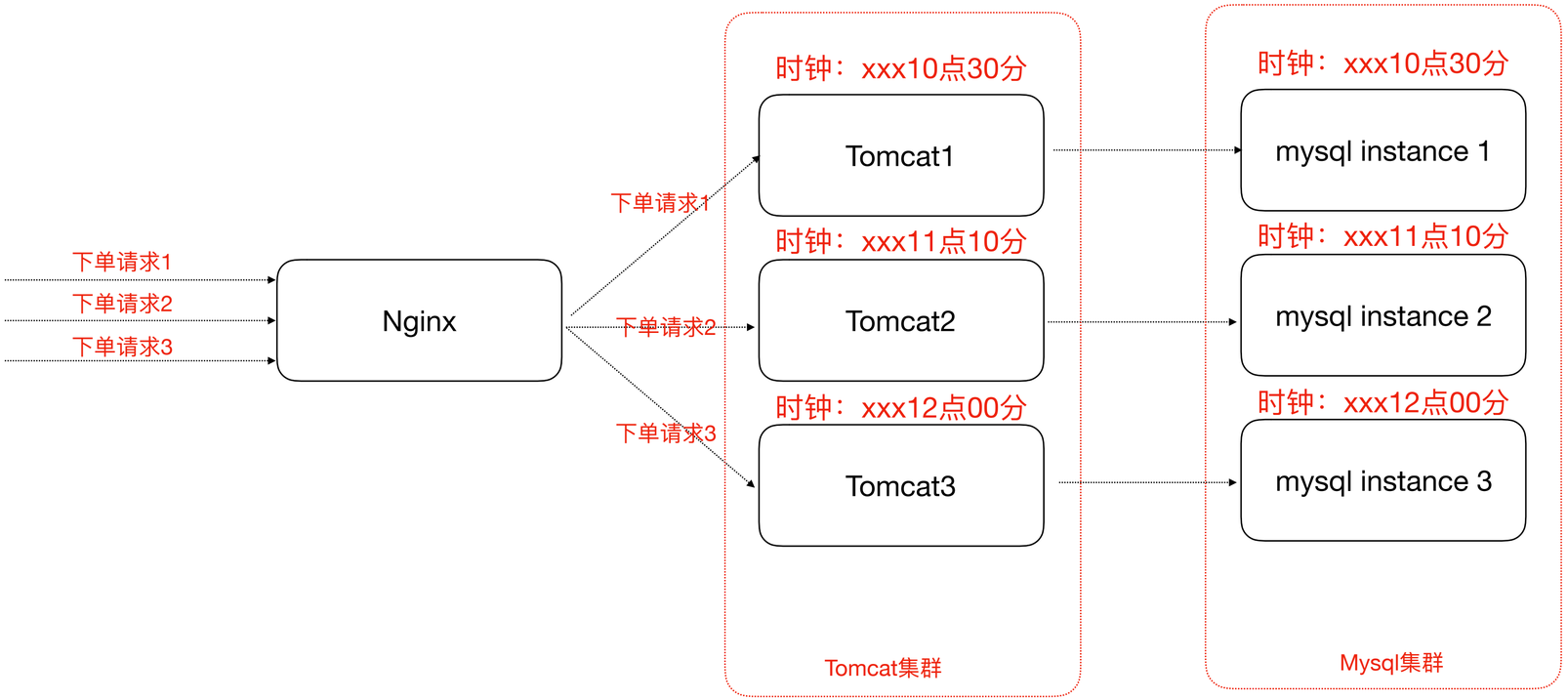
1）如前所述，每一台服务器负责一段，一致性哈希算法对于节点的增减都只需重定位环空间中的一小部分数据，具有较好的容错性和可扩展性。

但是，一致性哈希算法在服务节点太少时，容易因为节点分部不均匀而造成数据倾斜问题。例如系统中只有两台服务器，其环分布如下，节点2只能负责非常小的一段，大量的客户端请求落在了节点1上，这就是**数据倾斜问题**

2）为了解决这种数据倾斜问题，一致性哈希算法引入了虚拟节点机制，即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个此服务节点，称为虚拟节点。

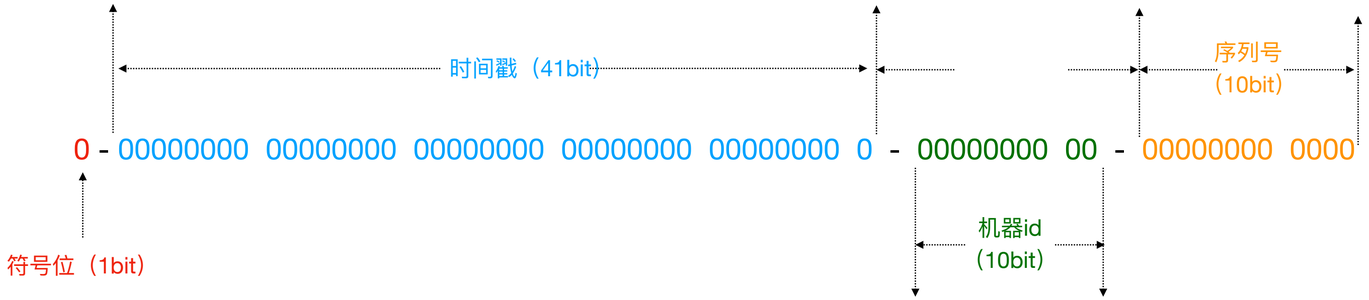
具体做法可以在服务器ip或主机名的后面增加编号来实现。比如，可以为每台服务器计算三个虚拟节点，于是可以分别计算 “节点1的ip#1”、“节点1的ip#2”、“节点1的ip#3”、“节点2的ip#1”、“节点2的ip#2”、“节点2的ip#3”的哈希值，于是形成六个虚拟节点，当客户端被路由到虚拟节点的时候其实是被路由到该虚拟节点所对应的真实节点

by 应癫　　集群时钟不一致问题示例



说明：并发的三个下单请求，几乎同时被处理，由于服务器时钟不一致，导致入库的下单时间却差别很大！数据混乱

by 应癫　　雪花算法原理示意



说明：

1）符号位：固定为0，二进制表示最高位是符号位，0代表正数，1代表负数。

2）时间戳：41个二进制数用来记录时间戳，表示某一个毫秒（毫秒级）

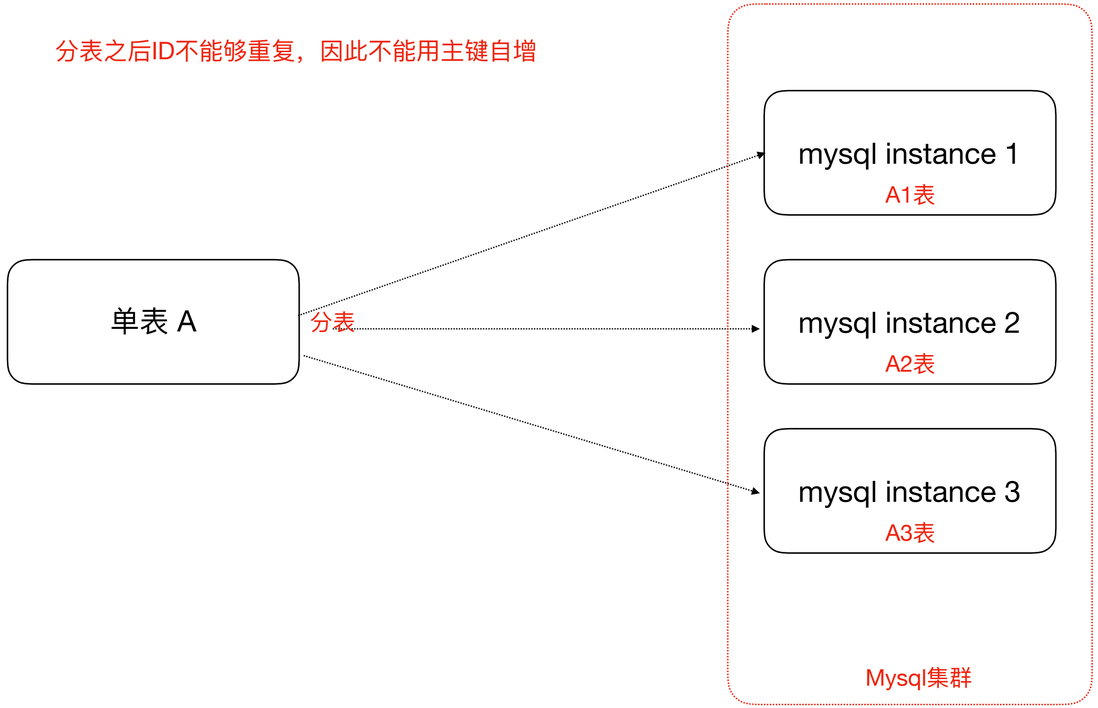
3）机器id：代表当前算法运行机器的id

4）序列号：12位，用来记录某个机器同一个毫秒内产生的不同序列号，代表同一个机器同一个毫秒可以产生的ID序号。

by 应癫

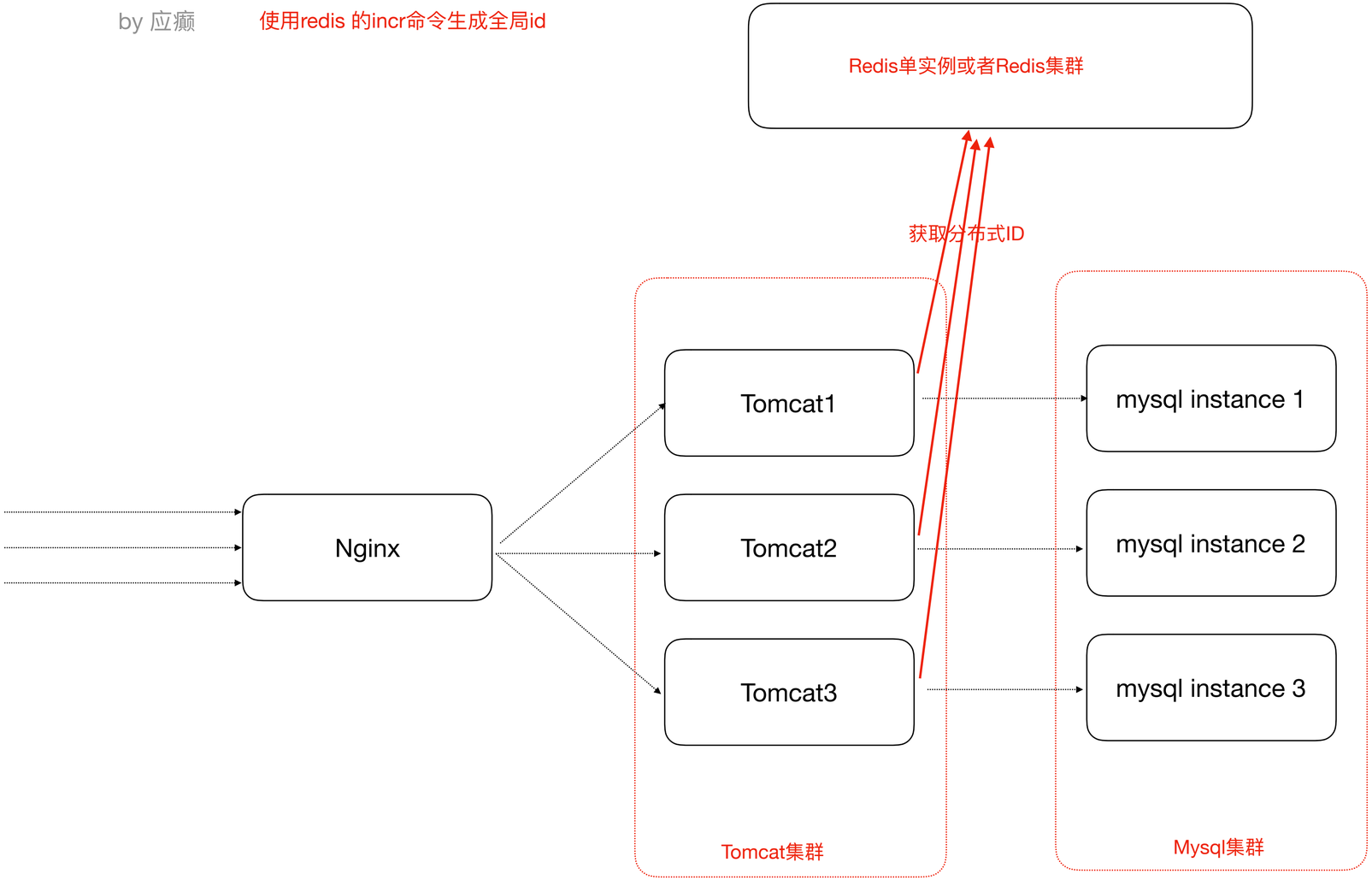
为什么需要使用分布式ID(分布式集群环境下的全局唯一ID)

分表之后ID不能够重复，因此不能用主键自增



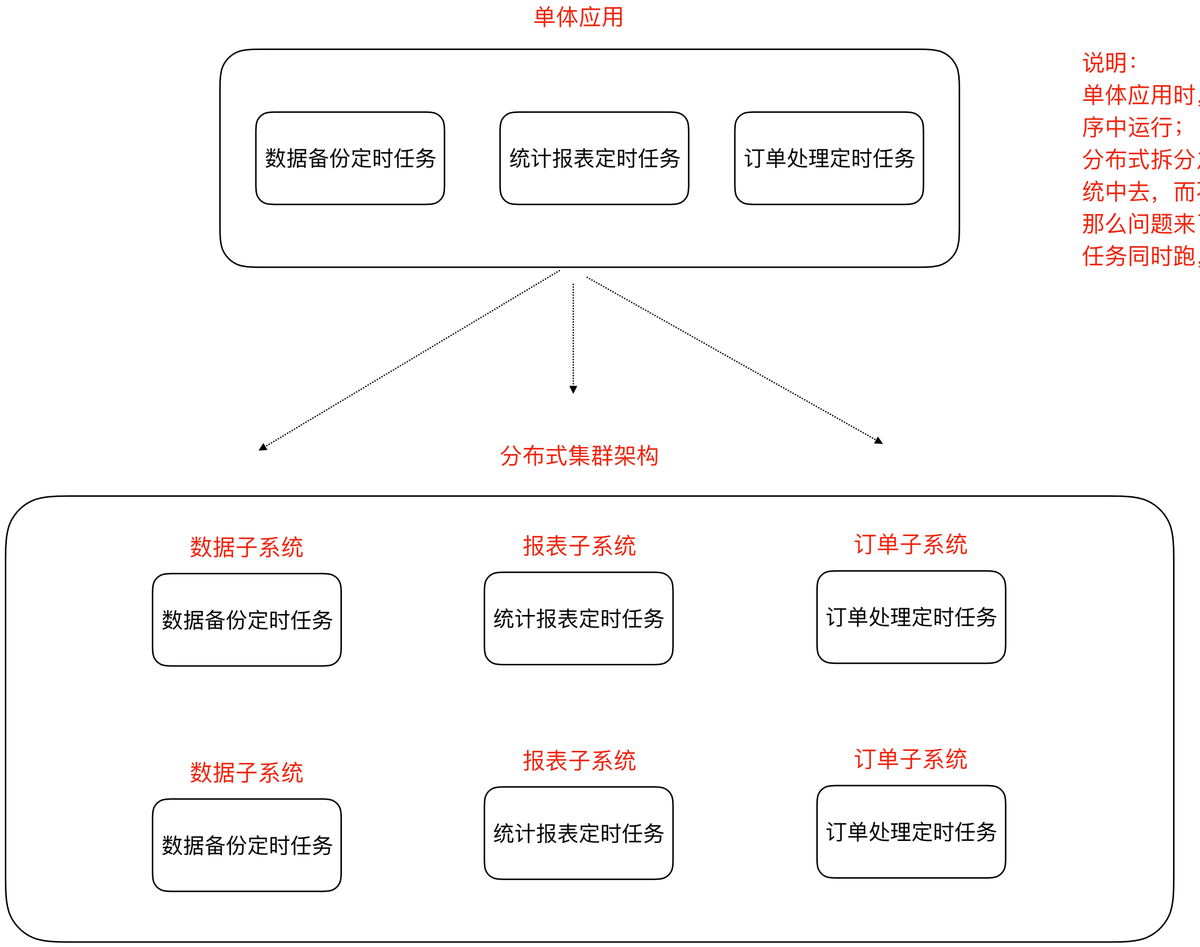
by 应癫

使用redis 的incr命令生成全局id



by 应癫

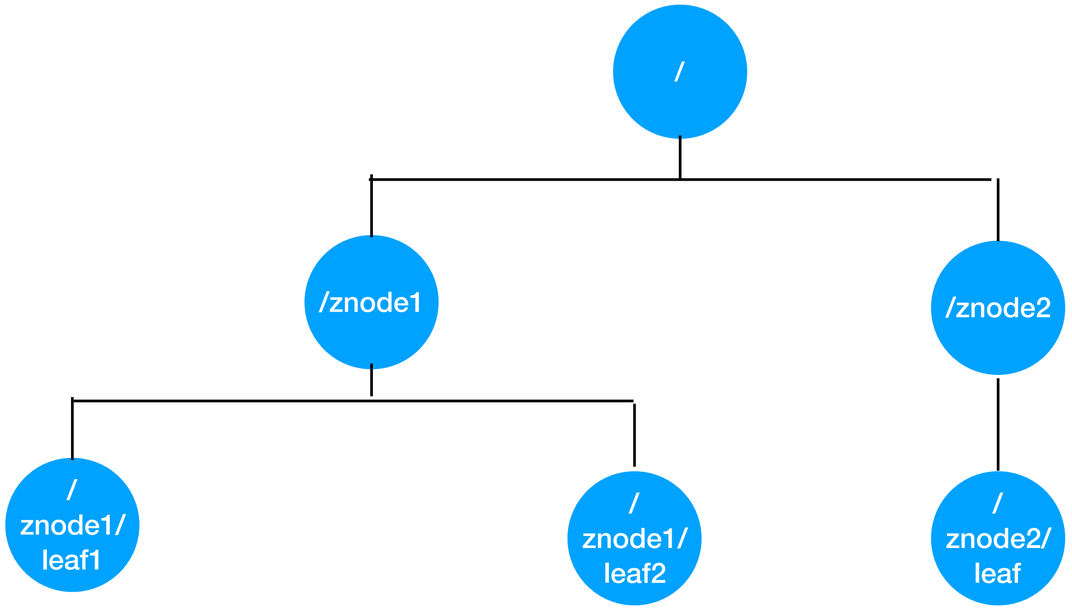
分布式调度



说明：
单体应用时，所有的定时任务都在一个服务器的一套程序中运行；
分布式拆分之后，不同的定时任务会拆分到不同的子系统中去，而不同的子系统又部署了多份；
那么问题来了，比如统计报表定时任务，如果两个定时任务同时跑，会不会重复处理?

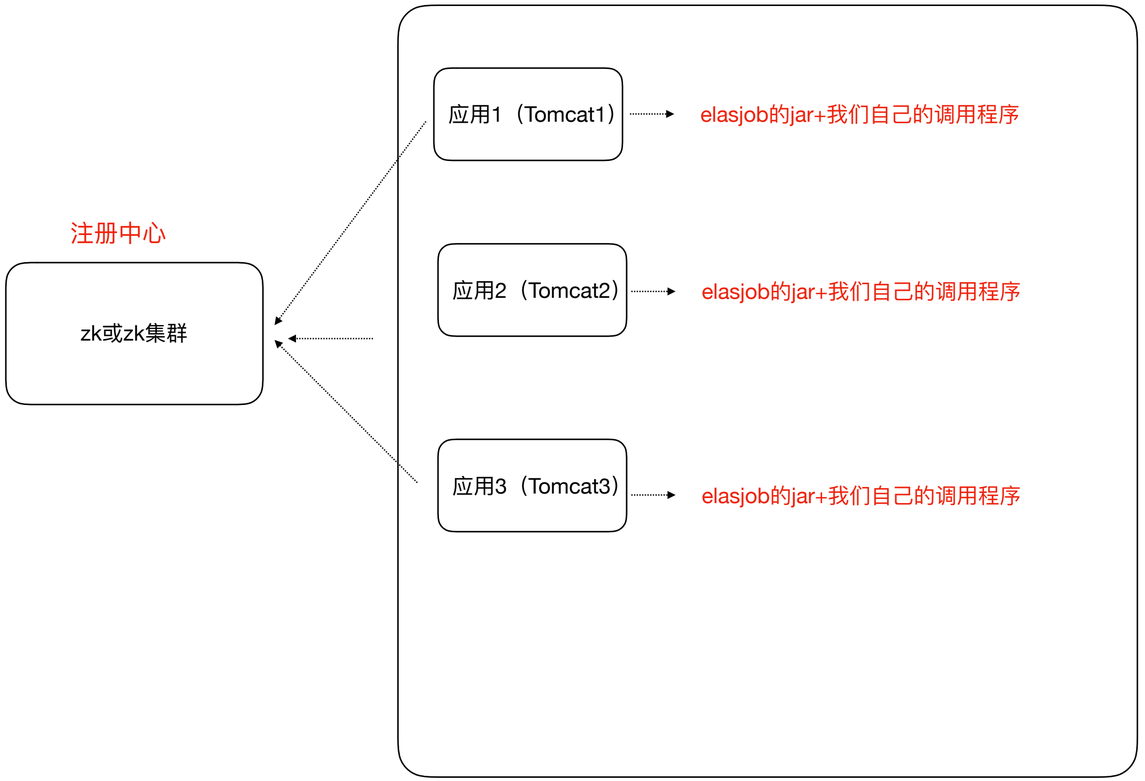
by 应癫

zookeeper树形结构



by 应癫

轻量级和去中心化

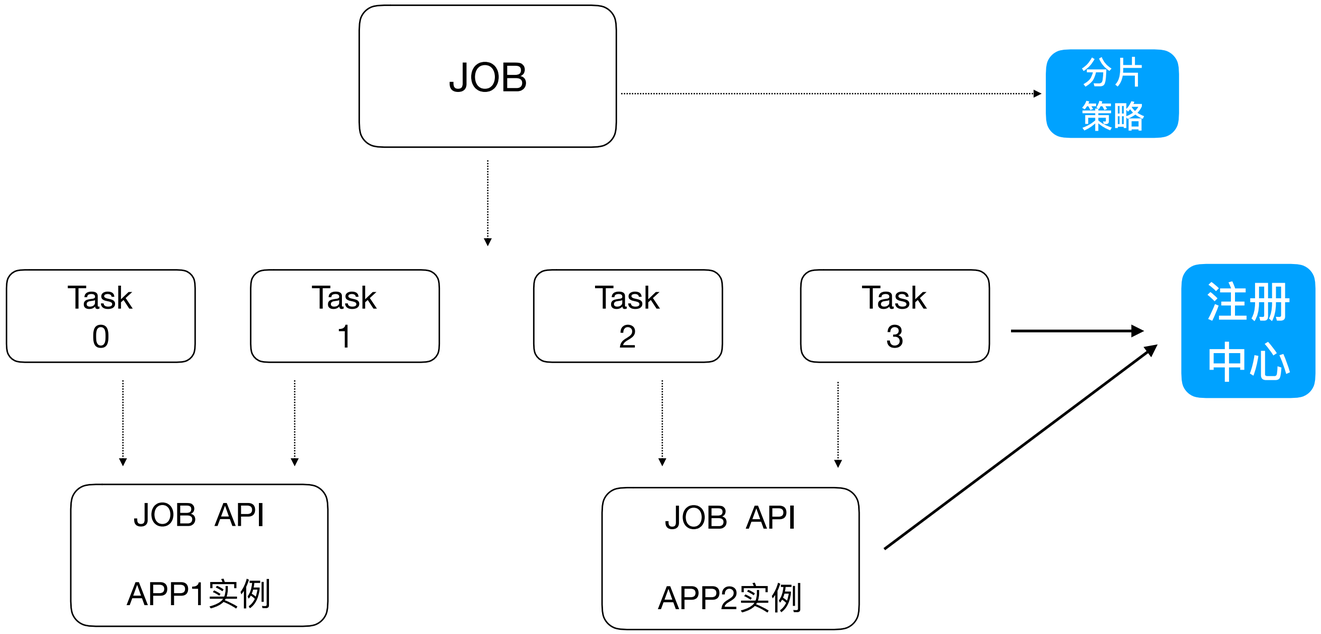


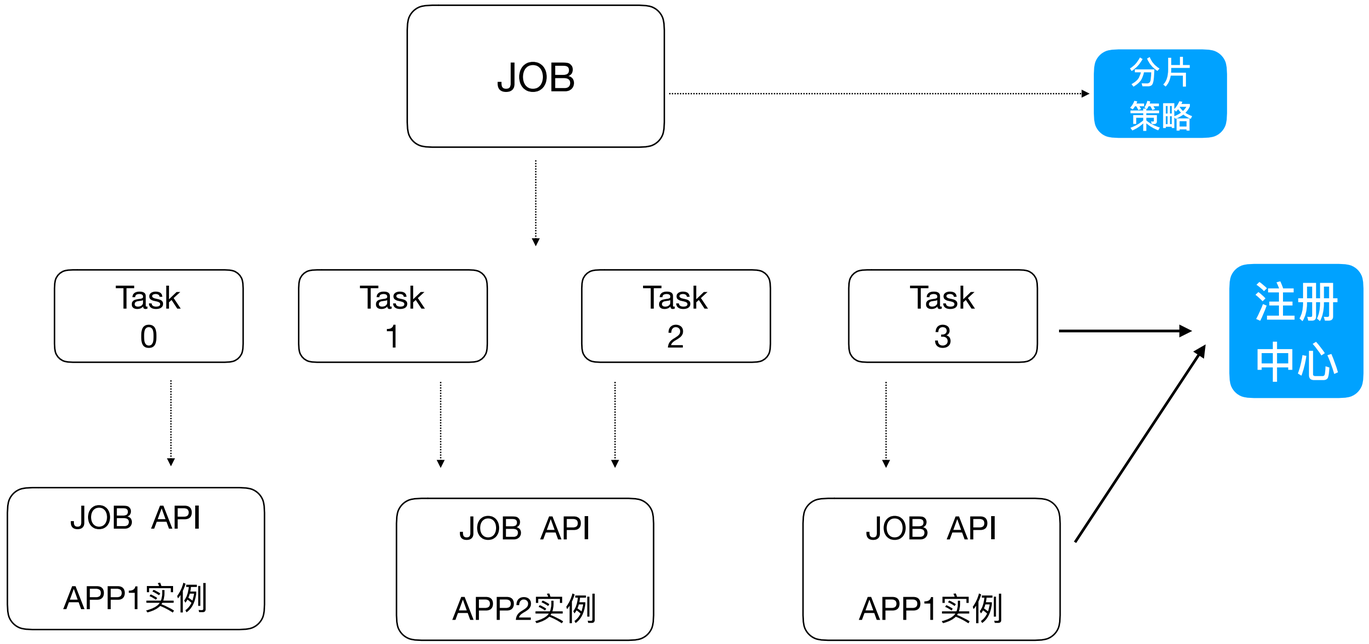
去中心化?

1）执行节点对等（程序和jar一样，唯一不一样的可能是分片）
2）定时调度自触发（没有中心调度节点分配）
3）服务自发现（过注册中心的服务发现）
4）主节点非固定

轻量级?

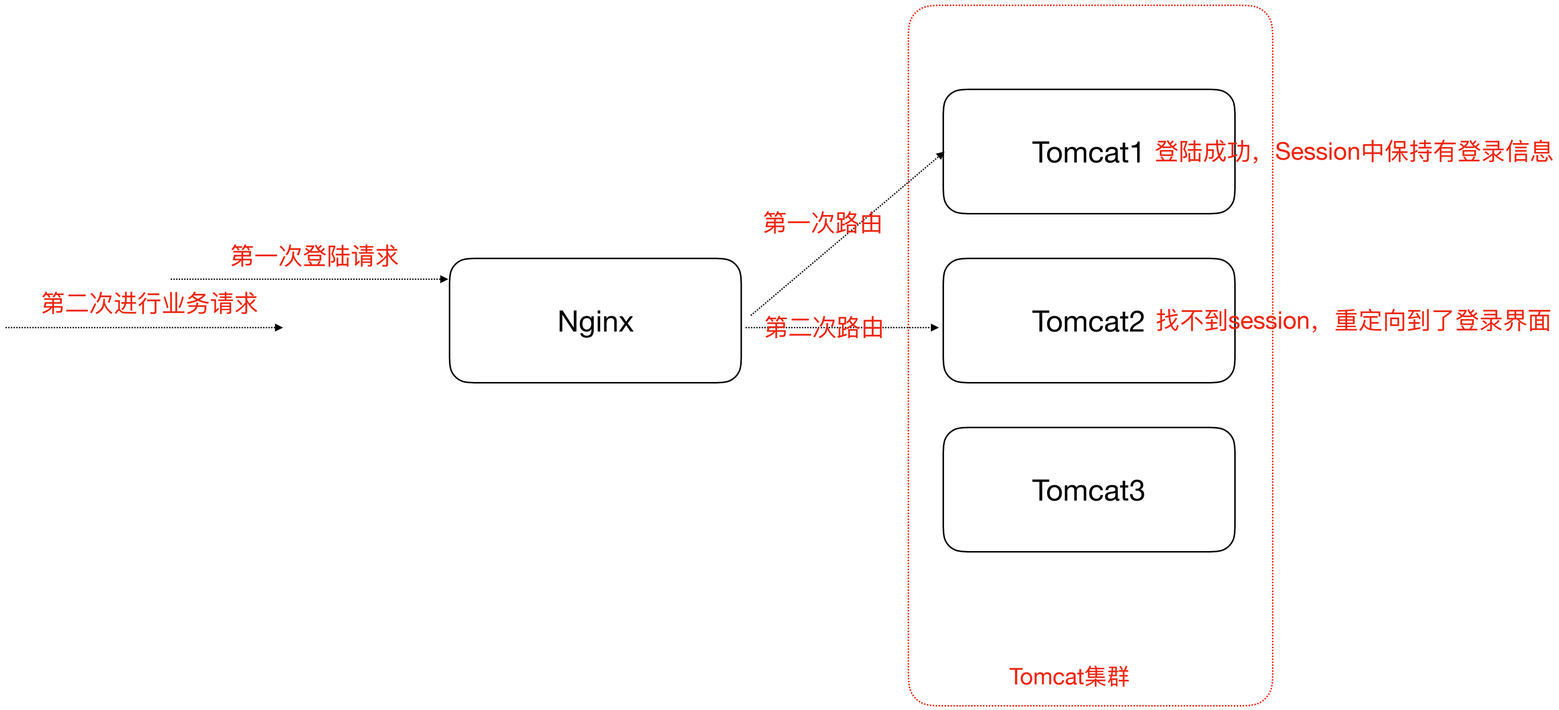
1）All in jar，必要依赖仅仅zookeeper
2）并非独立部署的中间件，就是jar程序


JOB
分片
策略
Task
0
Task
1
Task
2
Task
3
注册
中心
JOB API
APP1实例
JOB API
APP2实例


JOB
分片
策略
Task
0
Task
1
Task
2
Task
3
注册
中心
JOB API
APP1实例
JOB API
APP2实例
JOB API
APP1实例

by 应癫

Session共享问题



by 应癫　　Session丢失问题分析（重复登录）

第一次请求 http://111.229.248.243/LoginProject/login/toLogin

转发到login.jsp登录界面

8080 服务器

输入用户名和密码
发起第二次请求 http://111.229.248.243/LoginProject/login/loginSystem

8081 服务器

用户名密码正确（并且存入8081的Session）
重定向到Handler：/demo/result

但是这个重定向请求被负载到8080服务器
拦截器判断session中没有用户信息
拦截！重定向到登录页面

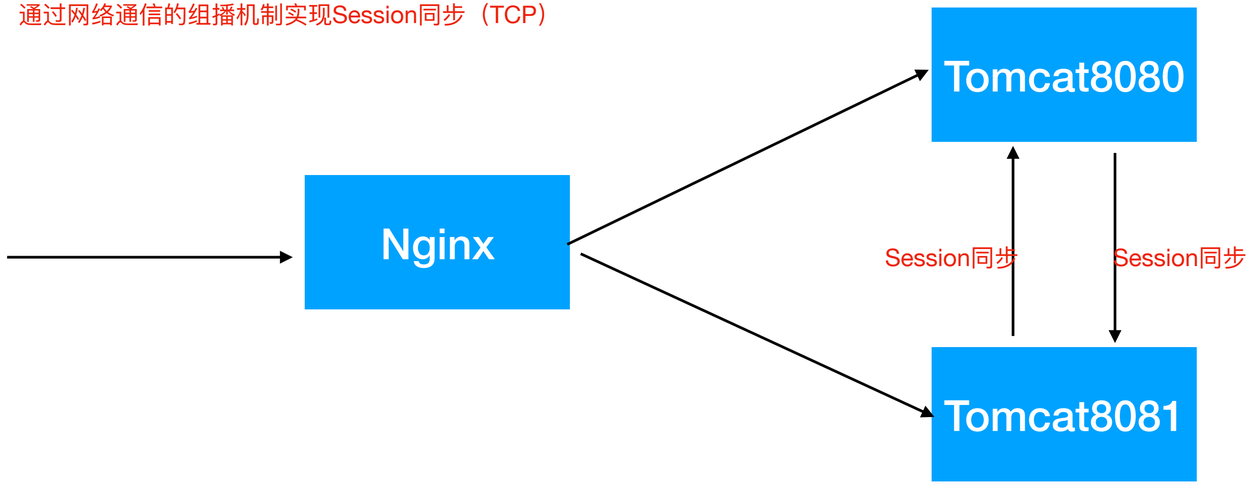
8080 服务器

这个重定向到登录页面的请求再次被负载到8081
让再次登录

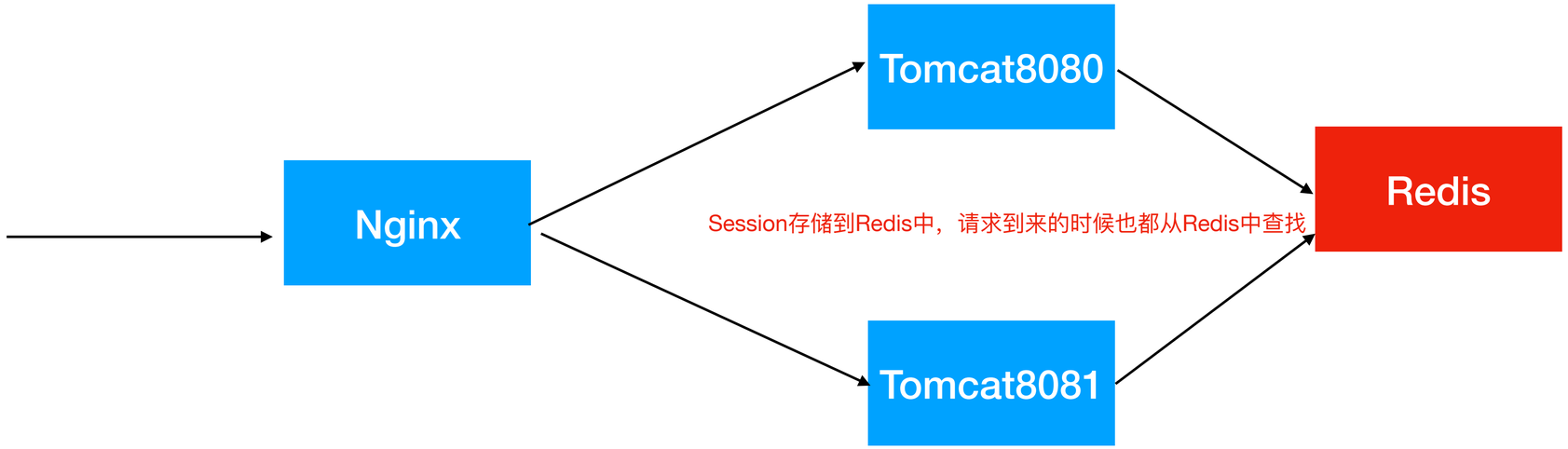
8081 服务器

by 应癫　　Tomcat的Session复制

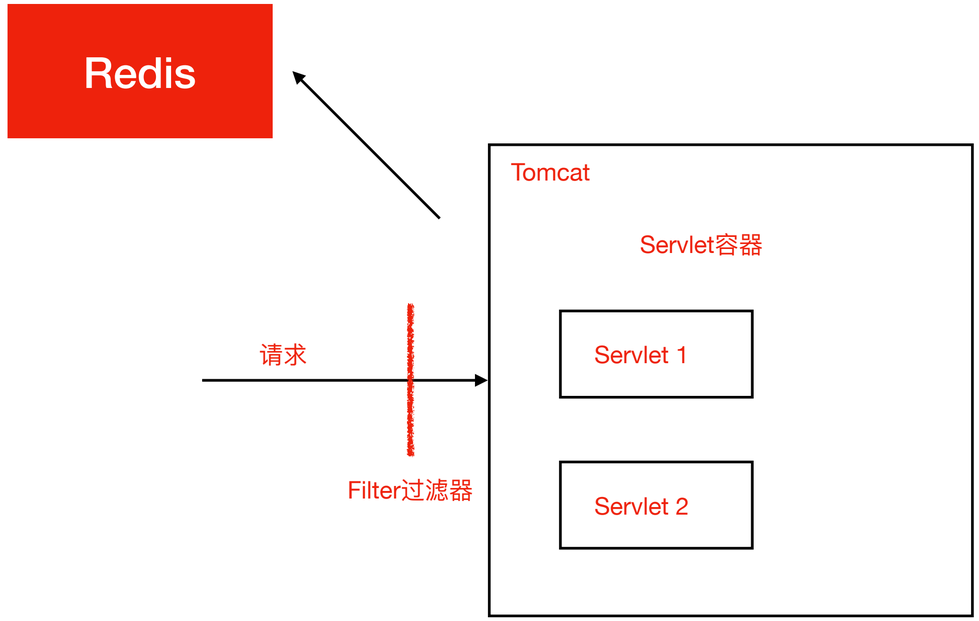
通过网络通信的组播机制实现Session同步（TCP）



by 应癫　　基于缓存中间件实现Session共享



by 应癫　　Spring Session原理示意



说明：
请求通过tomcat到达servlet容器的时候，通过过滤器对请求做了一次封装，如果没有过滤器，serlvet就会从tomcat中获取Session
有了过滤器之后，取出来的Session就是Redis中的Session，有的话就从Redis中获取，没有的话就创建并提交到Redis中去