



分布式基础篇

更多精品资源www.cx1314.cn

程序源码论坛-1024，网址 www.cx1314.cn 仅分享最流行最优质的IT资源！

不同于其他论坛平台，这里只有精品、稀有资源，已泛滥、已过时、垃圾资源不录入！

Java,前端,python,人工智能,大数据,云计算...持续更新资源-最新完整且均不加密、、、

活动线报，宅男福利，最新大片...

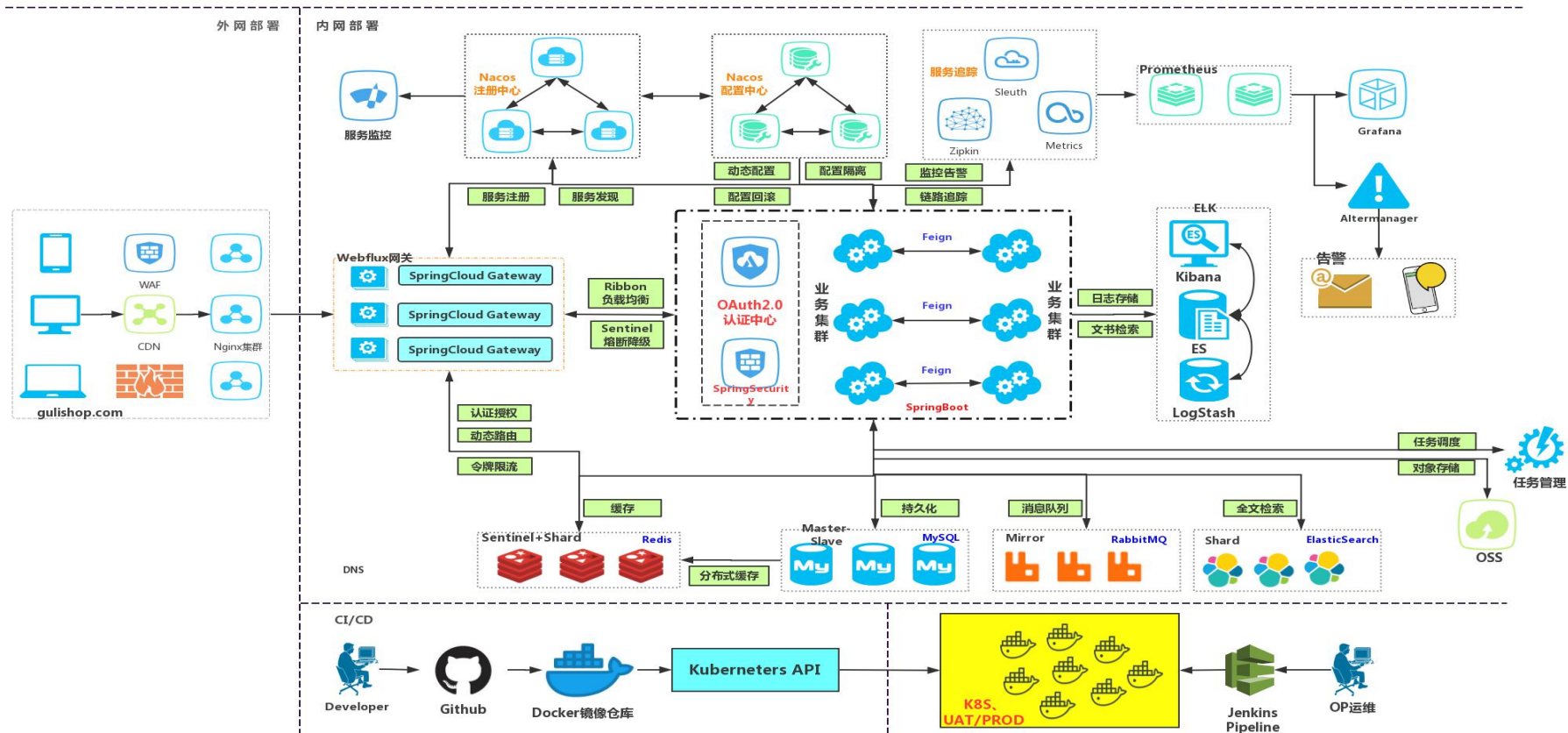
程序员的新大陆-更新最快的IT资源社区！开发者必备平台！

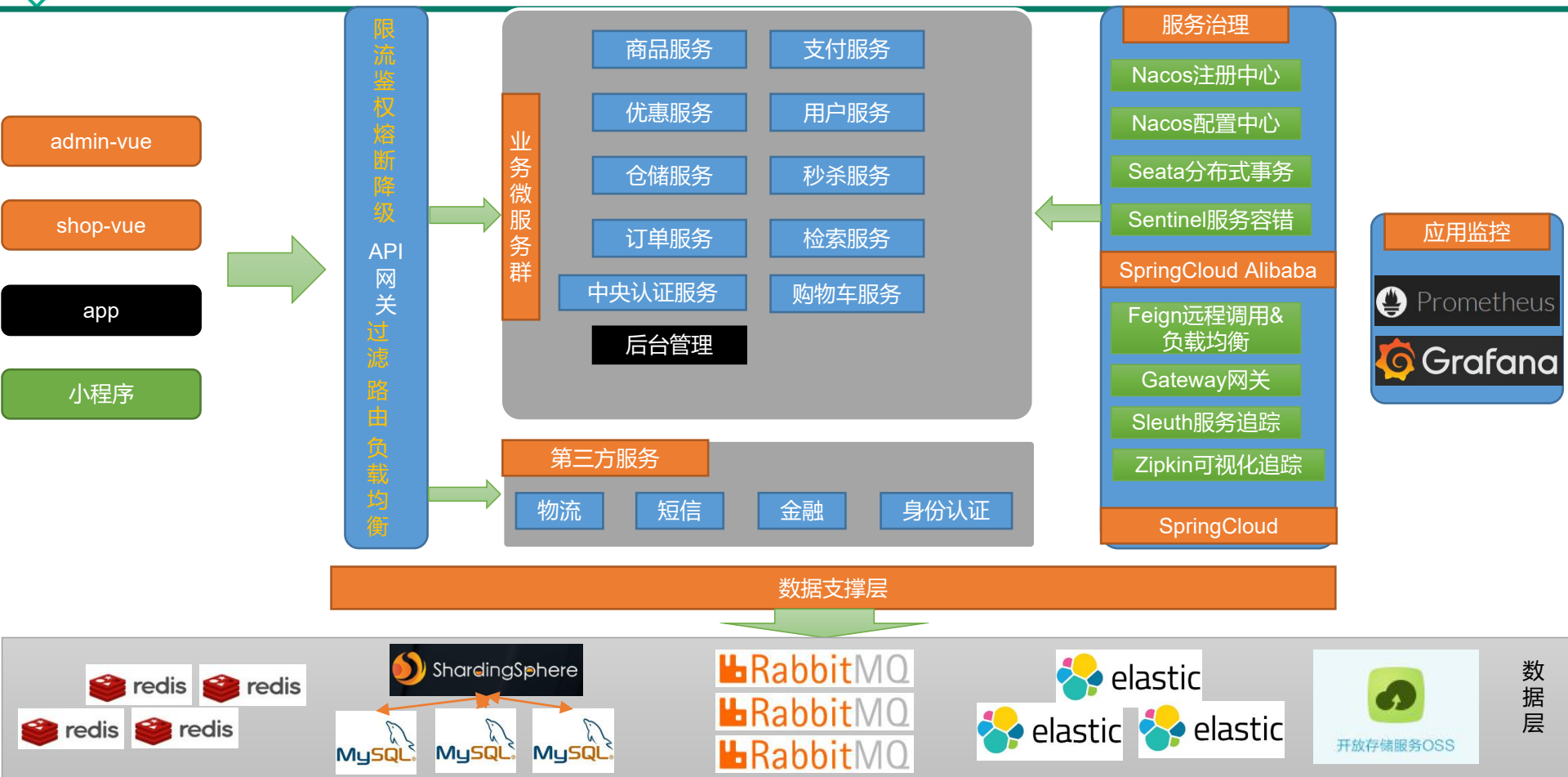
欢迎访问：www.cx1314.cn 百度搜索-> 程序源码论坛

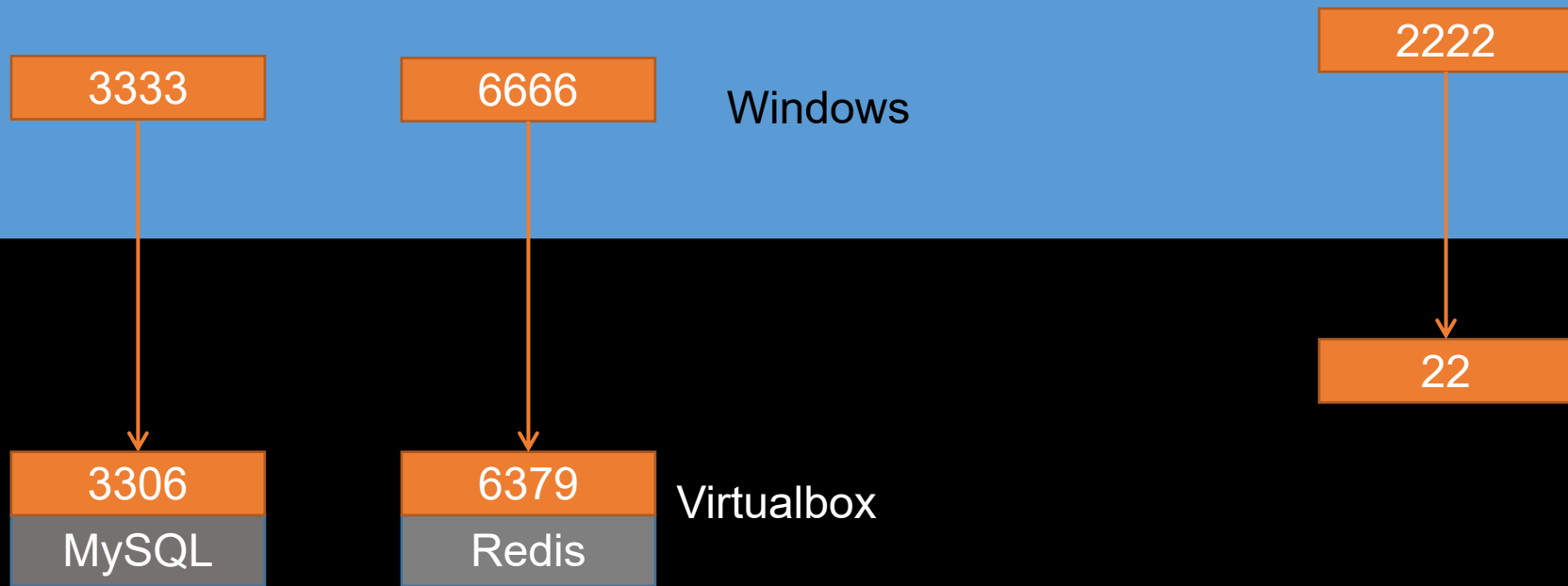
持续更新中！！！！ 最新资源，全网首发！~~~



谷粒商城-微服务架构图



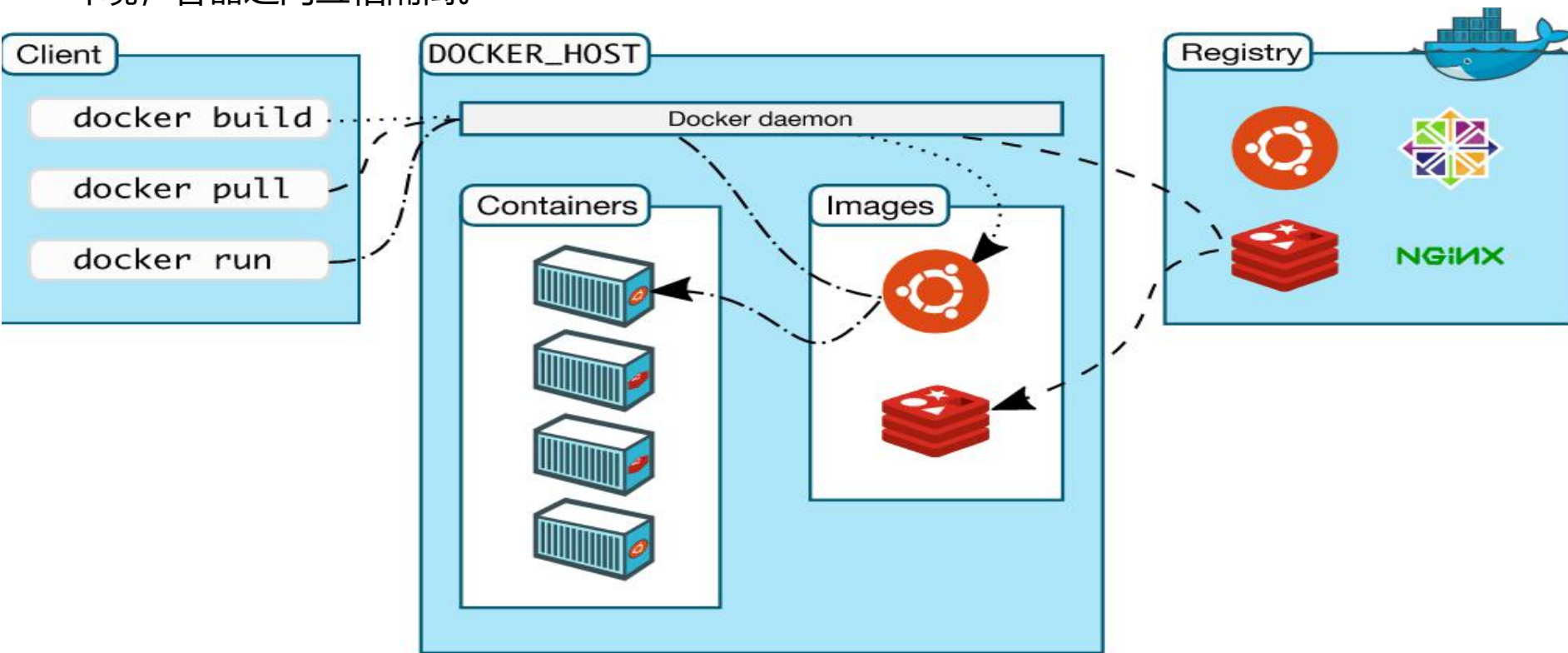






Docker

虚拟化容器技术。Docker基于镜像，可以秒级启动各种容器。每一种容器都是一个完整的运行环境，容器之间互相隔离。





linux

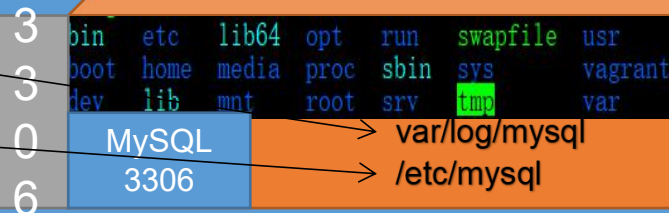
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
swapfile
sys
tmp
usr
var

/mydata/mysql/log

/mydata/mysql/conf

```
docker run -p 3306:3306 --name mysql \
-v /mydata/mysql/log:/var/log/mysql \
-v /mydata/mysql/data:/var/lib/mysql \
-v /mydata/mysql/conf:/etc/mysql \
-e MYSQL_ROOT_PASSWORD=root \
-d mysql:5.7
```

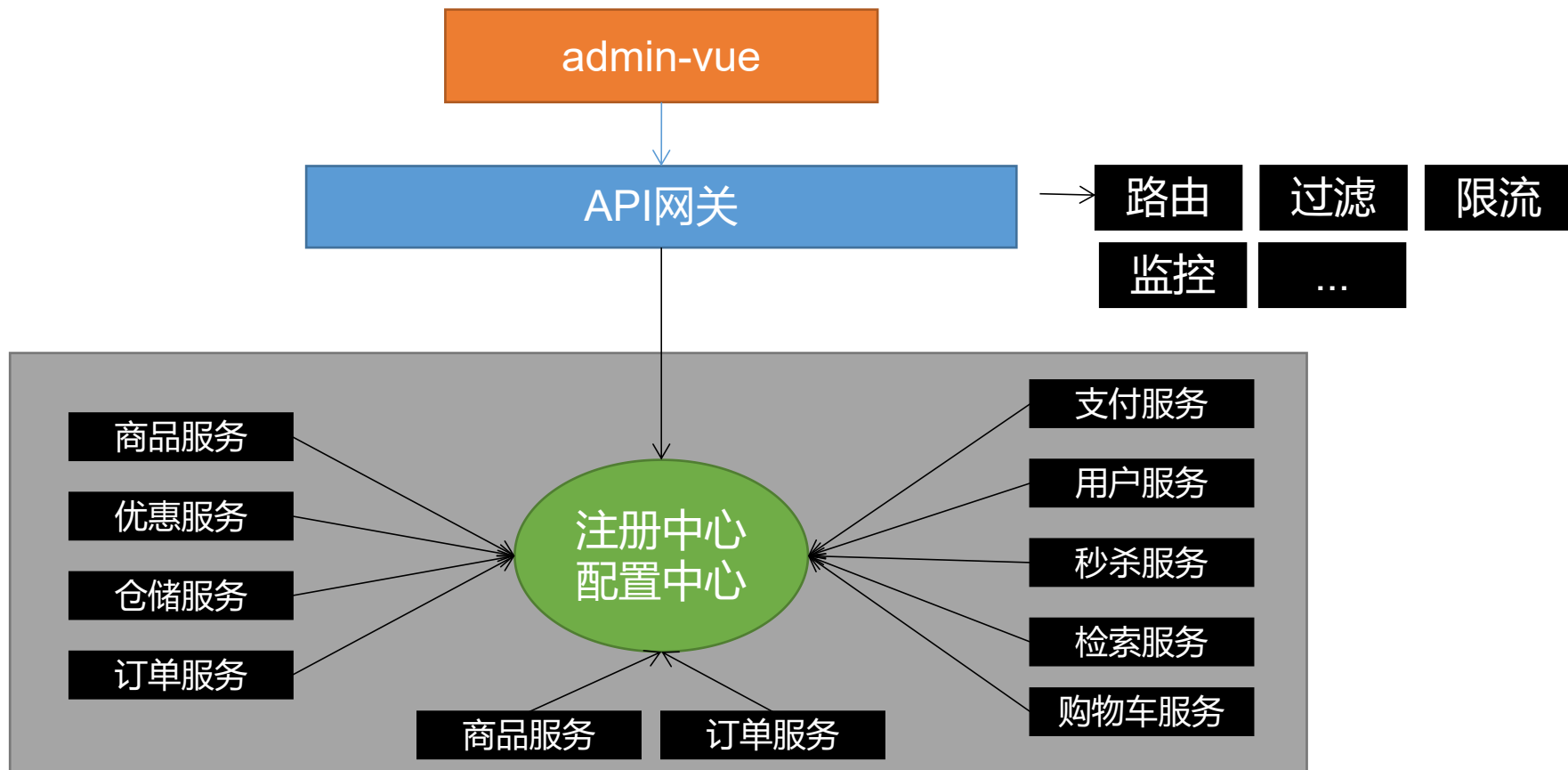
MySQL容器

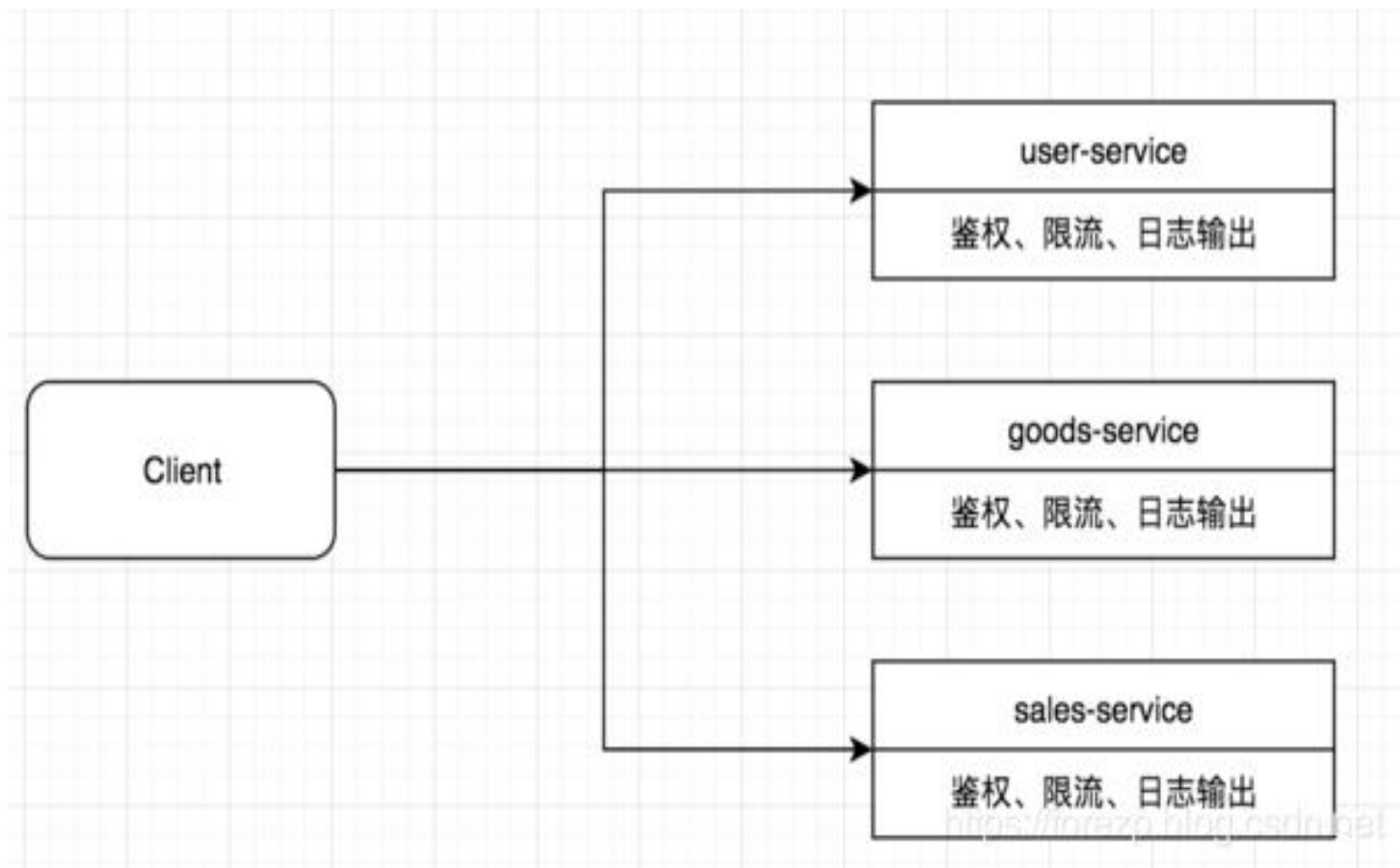


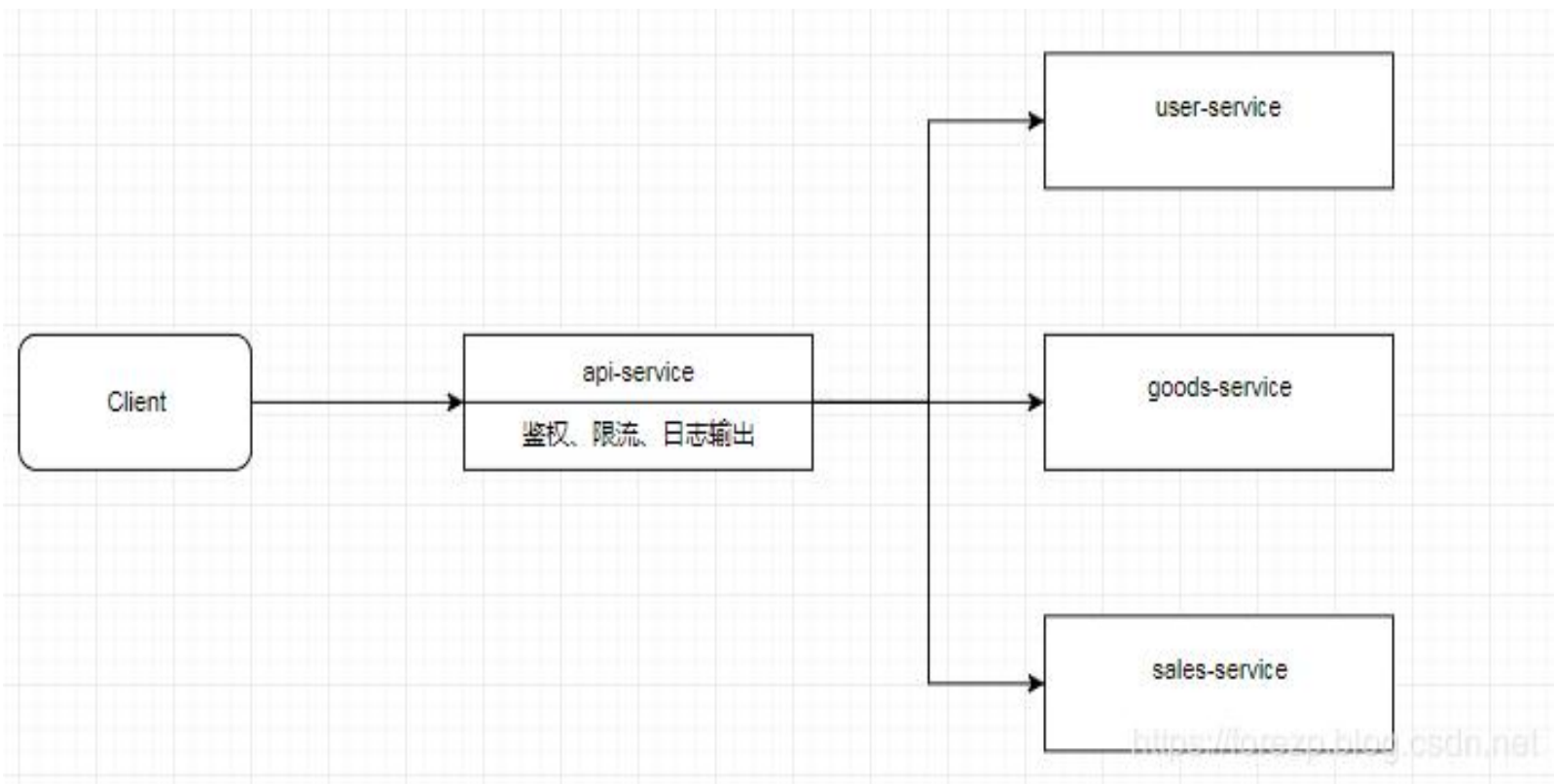
Redis容器



docker









JavaScript
es6,7,8...

框架
JQuery、Vue、
React

工具
webstorm,vscode

项目构建
webpack,gulp

依赖管理
npm

Java
jdk8,9,10,11...

框架
Spring、
SpringMVC

工具
idea,eclipse

项目构建
maven,gradle

依赖管理
maven

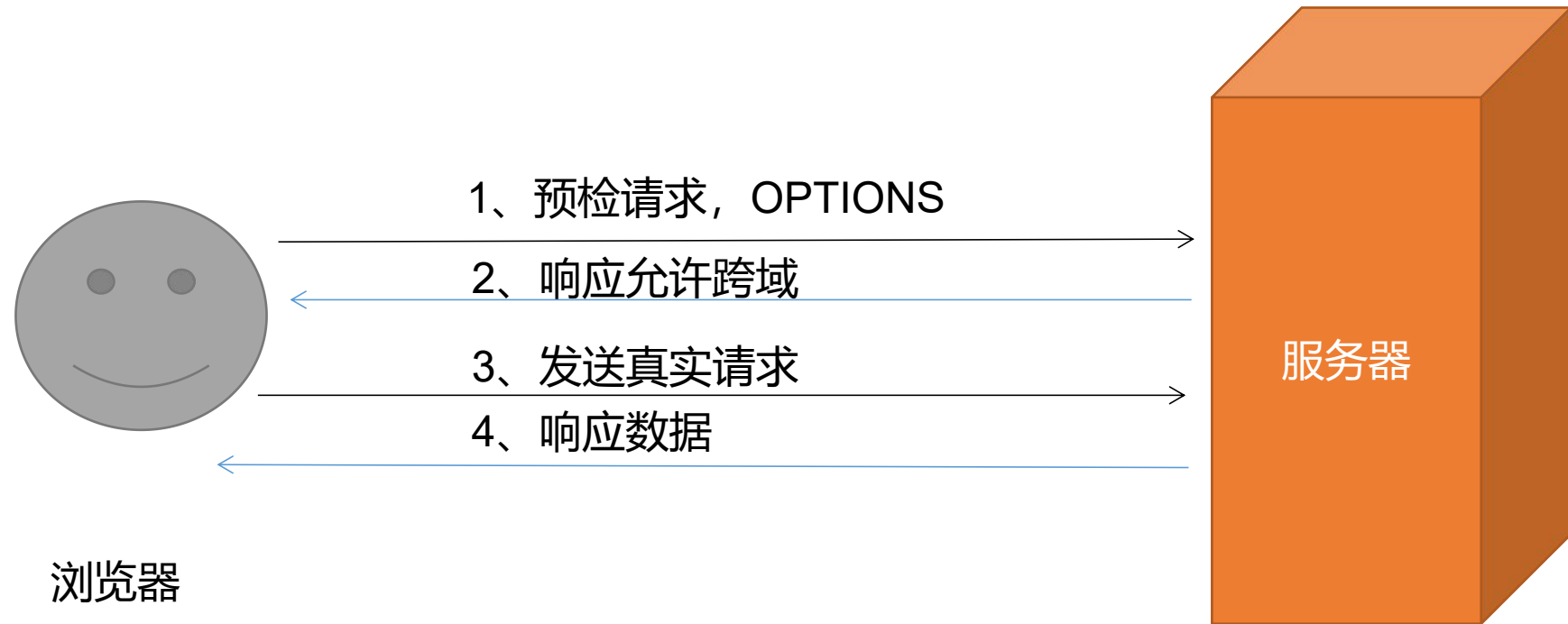


- **跨域**：指的是浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的，是**浏览器对javascript施加的安全限制**。
- **同源策略**：是指协议，域名，端口都要相同，其中有一个不同都会产生跨域；

URL	说明	是否允许通信
http://www.a.com/a.js http://www.a.com/b.js	同一域名下	允许
http://www.a.com/lab/a.js http://www.a.com/script/b.js	同一域名下不同文件夹	允许
http://www.a.com:8000/a.js http://www.a.com/b.js	同一域名，不同端口	不允许
http://www.a.com/a.js https://www.a.com/b.js	同一域名，不同协议	不允许
http://www.a.com/a.js http://70.32.92.74/b.js	域名和域名对应ip	不允许
http://www.a.com/a.js http://script.a.com/b.js	主域相同，子域不同	不允许
http://www.a.com/a.js http://a.com/b.js	同一域名，不同二级域名（同上）	不允许（cookie这种情况下也不允许访问）
http://www.cnblogs.com/a.js http://www.a.com/b.js	不同域名	不允许

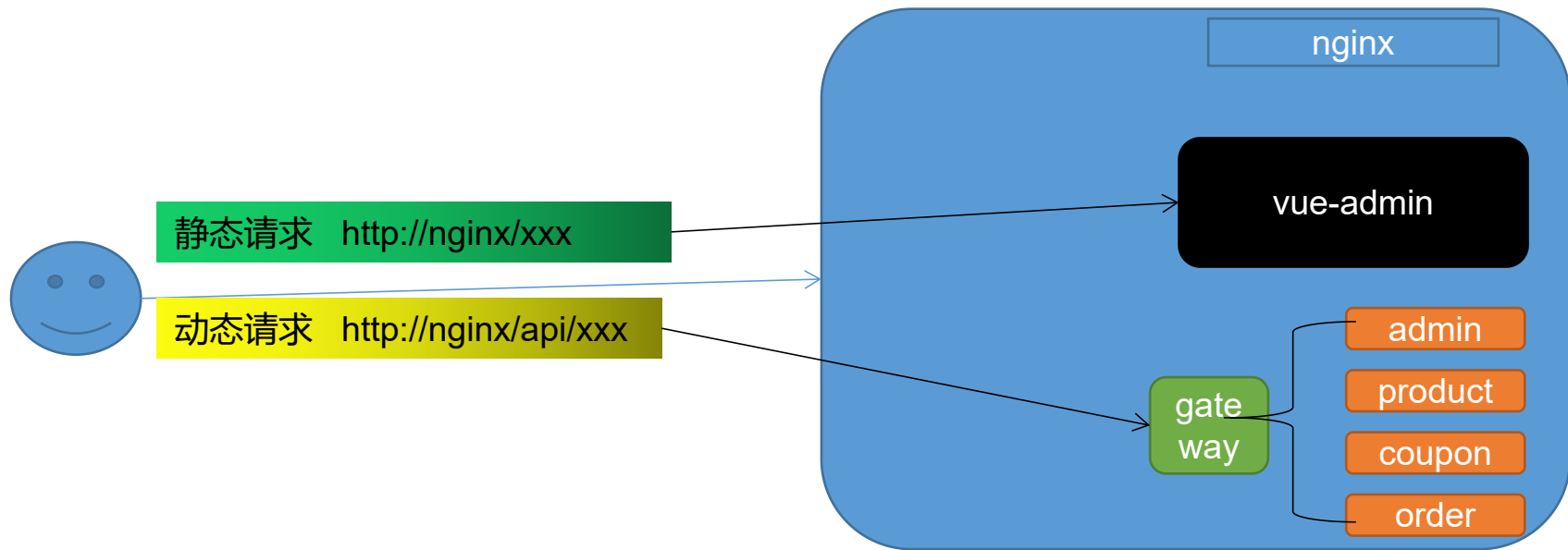


非简单请求（PUT、DELETE）等，需要先发送预检请求



浏览器

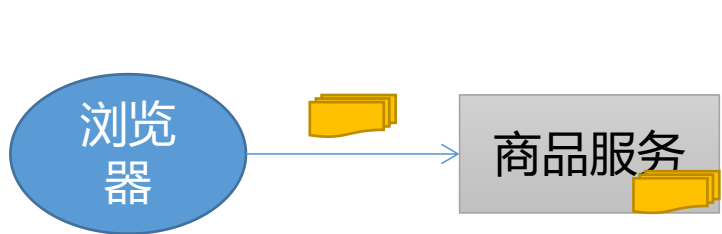
https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS



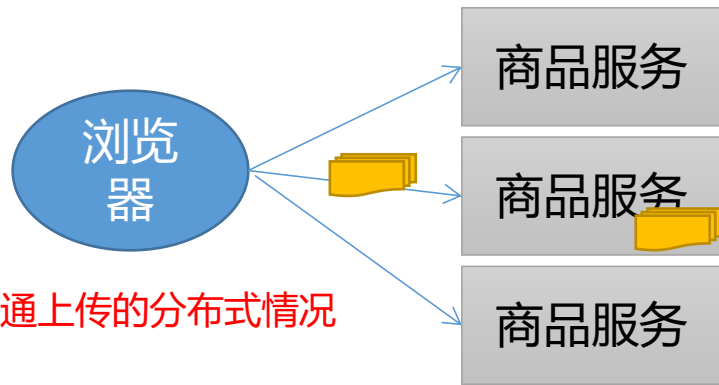


• 1、添加响应头

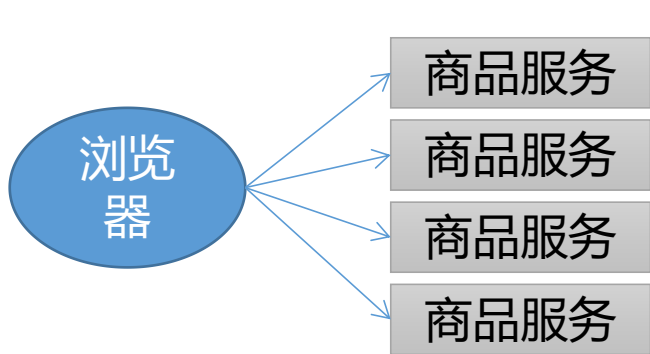
- Access-Control-Allow-Origin：支持哪些来源的请求跨域
- Access-Control-Allow-Methods：支持哪些方法跨域
- Access-Control-Allow-Credentials：跨域请求默认不包含cookie，设置为true可以包含cookie
- Access-Control-Expose-Headers：跨域请求暴露的字段
 - CORS请求时，XMLHttpRequest对象的getResponseHeader()方法只能拿到6个基本字段：Cache-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma。如果想拿到其他字段，就必须在Access-Control-Expose-Headers里面指定。
- Access-Control-Max-Age：表明该响应的有效时间为多少秒。在有效时间内，浏览器无须为同一请求再次发起预检请求。请注意，浏览器自身维护了一个最大有效时间，如果该首部字段的值超过了最大有效时间，将不会生效。

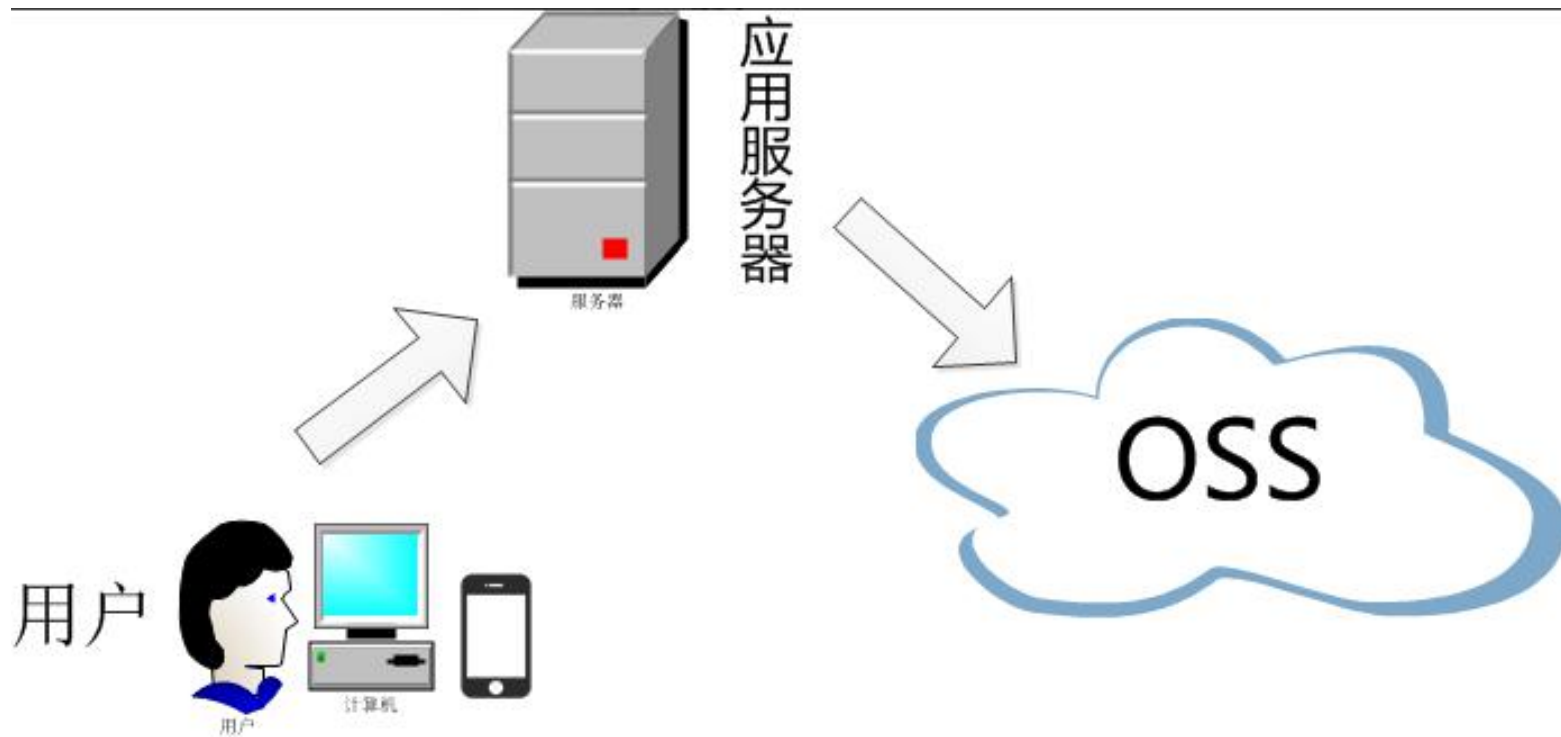


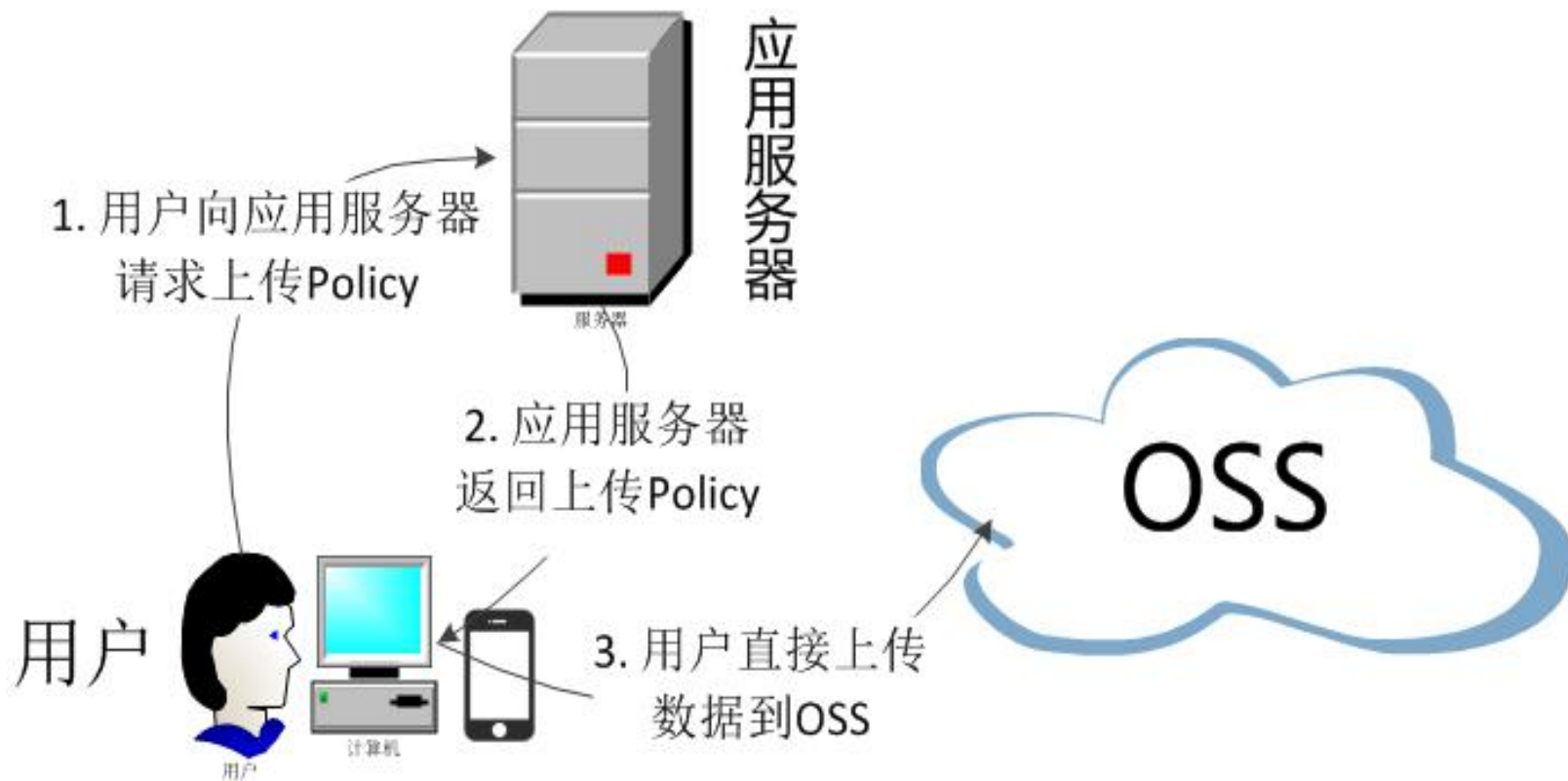
普通上传



普通上传的分布式情况









三级分类表

id	catelogName
1	手机
2	冰箱

属性分组表

id	groupName	catelogId
1	主体	1
2	屏幕	1

属性分组&属性关联表

id	attrGroupId	attrId
1	1	1
2	1	2

属性表

id	attrName	catelogId
1	内存	
2	像素	



到手价
1999
12-11.17)

抢券立减 1800 元

iPhone XS Max 256GB 深空灰色



Apple iPhone XS Max (A2104) 256GB 深空灰色 移动联通电信4

【抢券立减1800元!】iPhoneXS系列低至5599元! [详情请点击!](#)

11.11 全球好物节

京 东 价 **¥8999.00** 降价通知

优 惠 券 **满7000减1800**

促 销 **赠品** × 1 (赠完即止)

增值业务 回收限量加价350 套 3元1G

配 送 至 **陕西安康市紫阳县城关镇** 有货 支持 隔日达 | 自提 | 99元免差

由 **京东** 发货, 并提供售后服务. 16:00前下单, 预计**11月16日(周六)**送达

重 量 0.48kg

选择颜色



金色



深空灰色



银色

选择版本

64GB

256GB

512GB

购买方式

公开版

移动4G优先版

换修无忧年付版

换修无忧月付

购买方式

官方标配

移动优惠购

电信优惠购



product_attr_value: 商品属性值表

id	spuld	attrId	attrVal
1	1	2	3000万
2	1	1	3G;4G;5G

sku_sale_attr_value: sku销售属性值表

id	skuld	spuld	attrId	attrVal
1	1	1	3	6G
2	1	1	4	128G
3	2	1	3	4G
4	2	1	4	64G

属性表

id	attrName	cateId
1	网络	1
2	像素	1
3	内存	1
4	容量	1

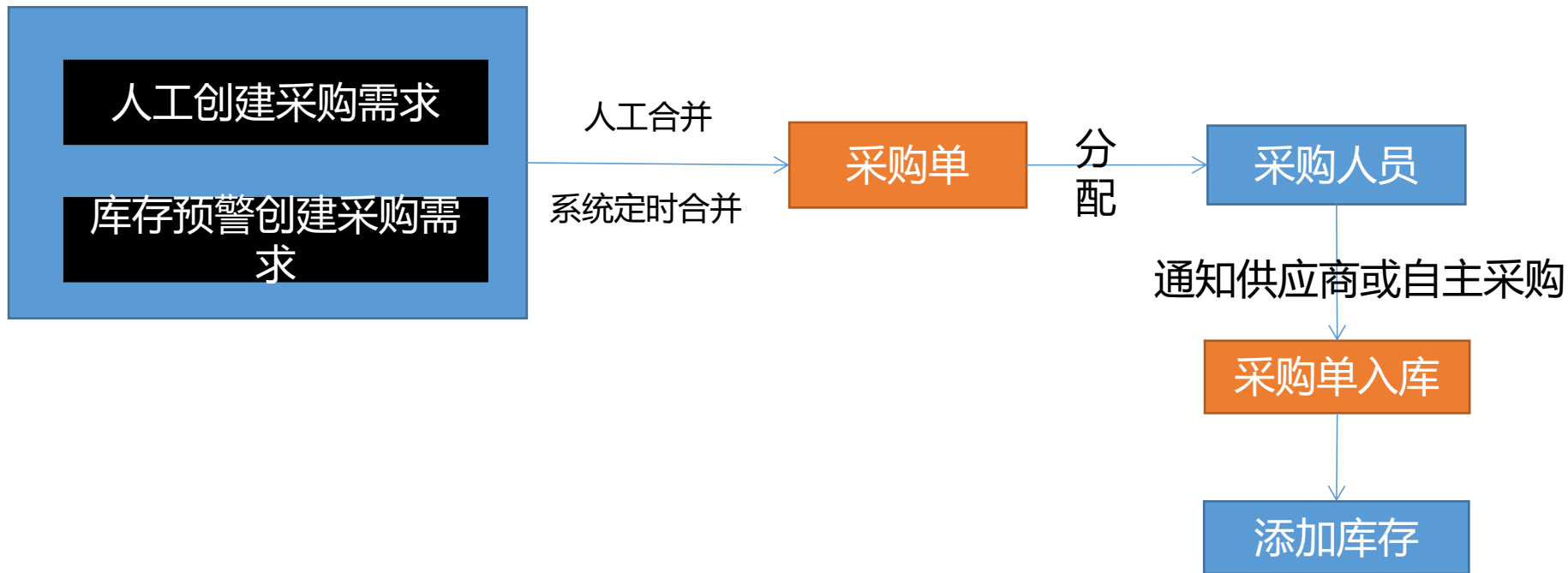


- ▶ 图书、音像、电子书刊
- ▶ 手机
- ▶ 大家电
- ▶ 数码
- ▶ 家用电器
- ▶ 家居家装
- ▶ 厨具
- ▶ 电脑办公
- ▶ 个护化妆
- ▶ 服饰内衣
- ▶ 钟表
- ▶ 鞋靴

<input type="checkbox"/>	分组id	组名	排序	描述	组图标
暂无数据					

共 0 条

10条/页





- **1、分布式基础概念**

- 微服务、注册中心、配置中心、远程调用、Feign、网关

- **2、基础开发**

- SpringBoot2.0、SpringCloud、Mybatis-Plus、Vue组件化、阿里云对象存储

- **3、环境**

- Vagrant、Linux、Docker、MySQL、Redis、逆向工程&人人开源

- **4、开发规范**

- 数据校验JSR303、全局异常处理、全局统一返回、全局跨域处理
- 枚举状态、业务状态码、VO与TO与PO划分、逻辑删除
- Lombok: @Data、@Slf4j



分布式高级篇

程序源码论坛-1024，网址 www.cx1314.cn 仅分享最流行最优质的IT资源！

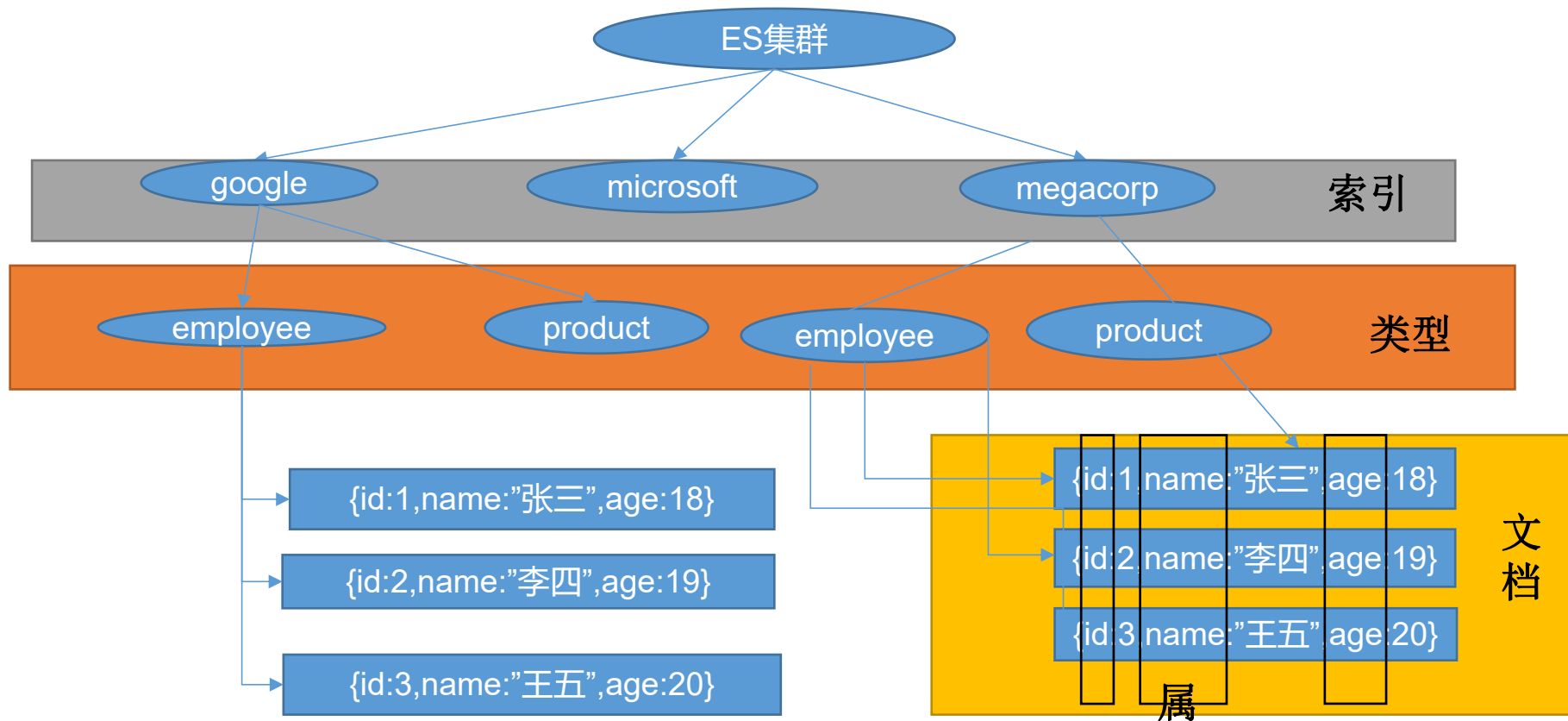
不同于其他论坛平台，这里只有精品、稀有资源，已泛滥、已过时、垃圾资源不录入！

Java,前端,python,人工智能,大数据,云计算...持续更新资源-最新完整且均不加密、、、

活动线报，宅男福利，最新大片...

程序员的新大陆-更新最快的IT资源社区！开发者必备平台！

欢迎访问：www.cx1314.cn 百度搜索-> 程序源码论坛





词	记录
红海	1,2,3,4,5
行动	1,2,3
探索	2,5
特别	3,5
记录篇	4
特工	5

分词：将整句分拆为单词

保存的记录

- 1-红海行动
- 2-探索红海行动
- 3-红海特别行动
- 4-红海记录篇
- 5-特工红海特别探索

检索：

- 1)、红海特工行动?
- 2)、红海行动?

相关性得分：



- 关系型数据库中两个数据表示是独立的，即使他们里面有相同名称的列也不影响使用，但ES中不是这样的。elasticsearch是基于Lucene开发的搜索引擎，而ES中不同type下名称相同的field最终在Lucene中的处理方式是一样的。
 - 两个不同type下的两个user_name，在ES同一个索引下其实被认为是同一个field，你必须在两个不同的type中定义相同的field映射。否则，不同type中的相同字段名称就会在处理中出现冲突的情况，导致Lucene处理效率下降。
 - 去掉type就是为了提高ES处理数据的效率。
- Elasticsearch 7.x
 - URL中的type参数为可选。比如，索引一个文档不再要求提供文档类型。
- Elasticsearch 8.x
 - 不再支持URL中的type参数。
- 解决：将索引从多类型迁移到单类型，每种类型文档一个独立索引



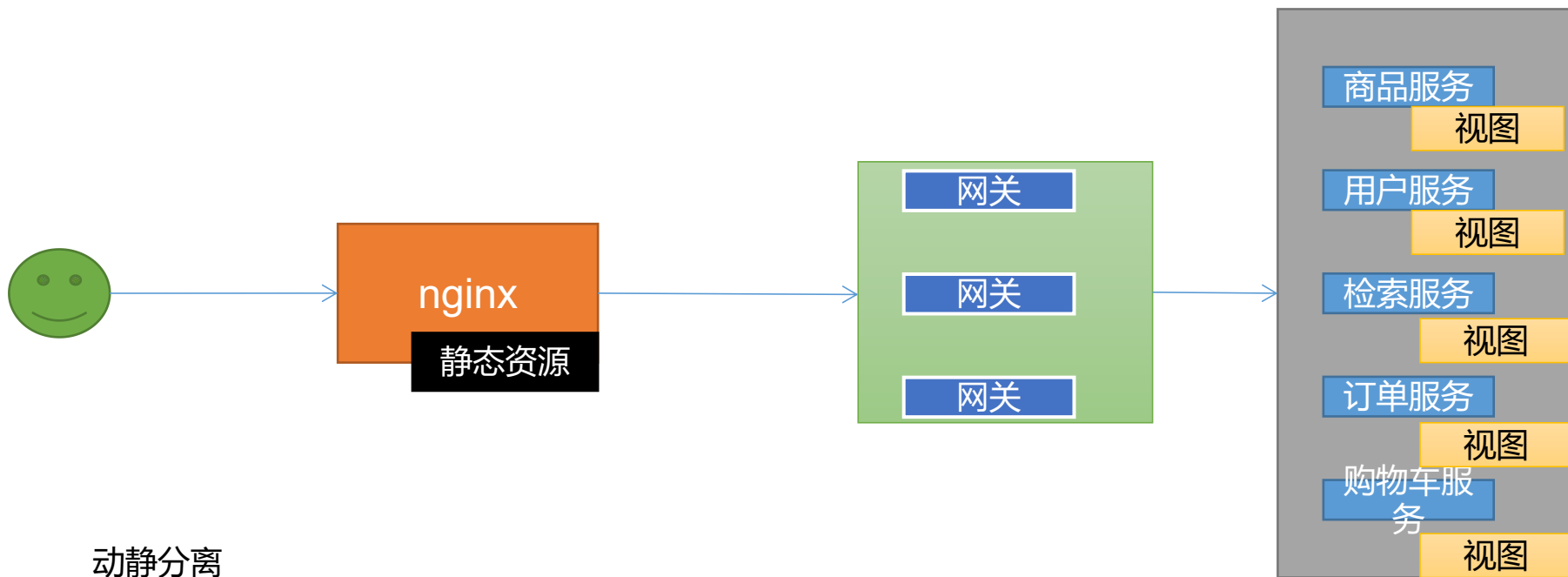
```
{
  "group" : "fans",
  "user" : [ ❶
    {
      "first" : "John",
      "last" : "Smith"
    },
    {
      "first" : "Alice",
      "last" : "White"
    }
  ]
}
```

扁平化

```
{
  "group" : "fans",
  "user.first" : [ "alice", "john" ],
  "user.last" : [ "smith", "white" ]
}
```

检索（实际并没有Alice Smith,但是会检索到）

```
{ "match": { "user.first": "Alice" } },
{ "match": { "user.last": "Smith" } }
```

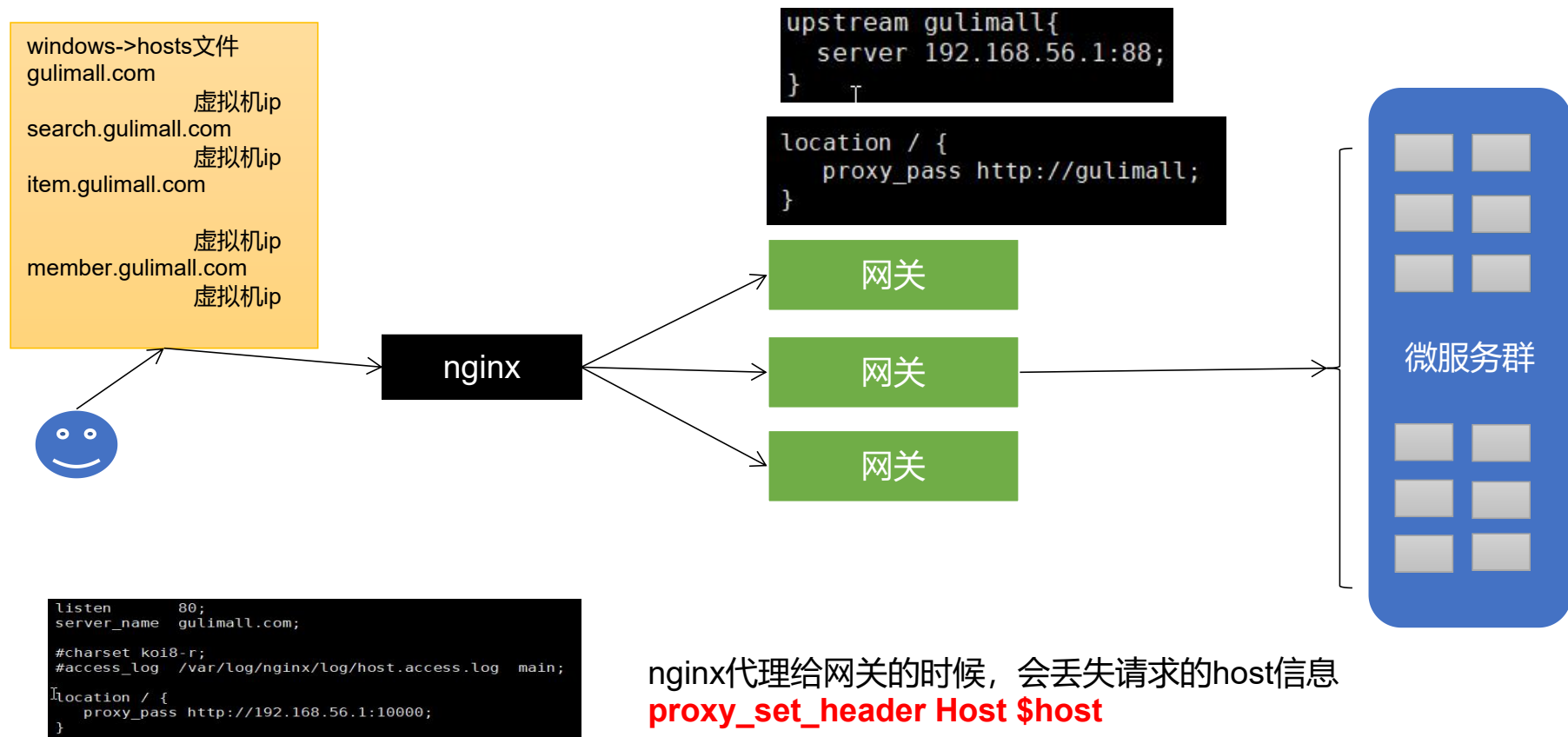


动静分离

静：图片，js、css等静态资源（以实际文件存在的方式）

动：服务器需要处理的请求

每一个微服务都可以独立部署、运行、升级
独立自治；技术，架构，业务



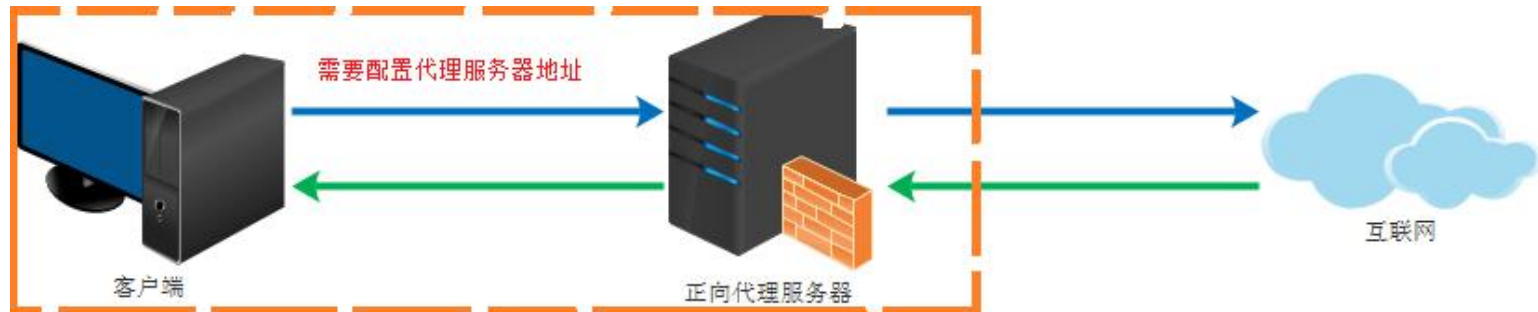
让nginx帮我们进行反向代理, 所有来自原gulimall.com的请求, 都转到商品服务



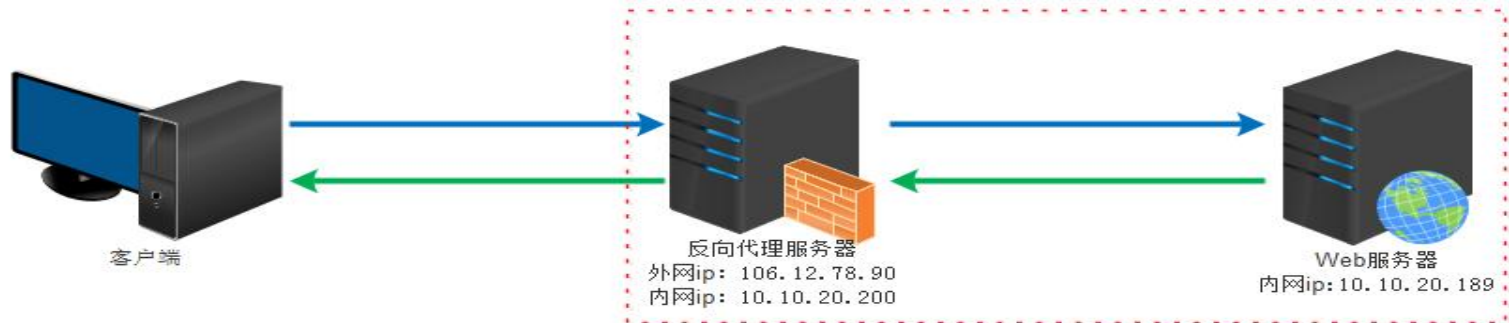
- 请求接口 gulimall.com
- 请求页面 gulimall.com
- nginx直接代理给网关，网关判断
 - 如果/api/****，转交给对应的服务器
 - 如果是 满足域名，转交给对应的服务

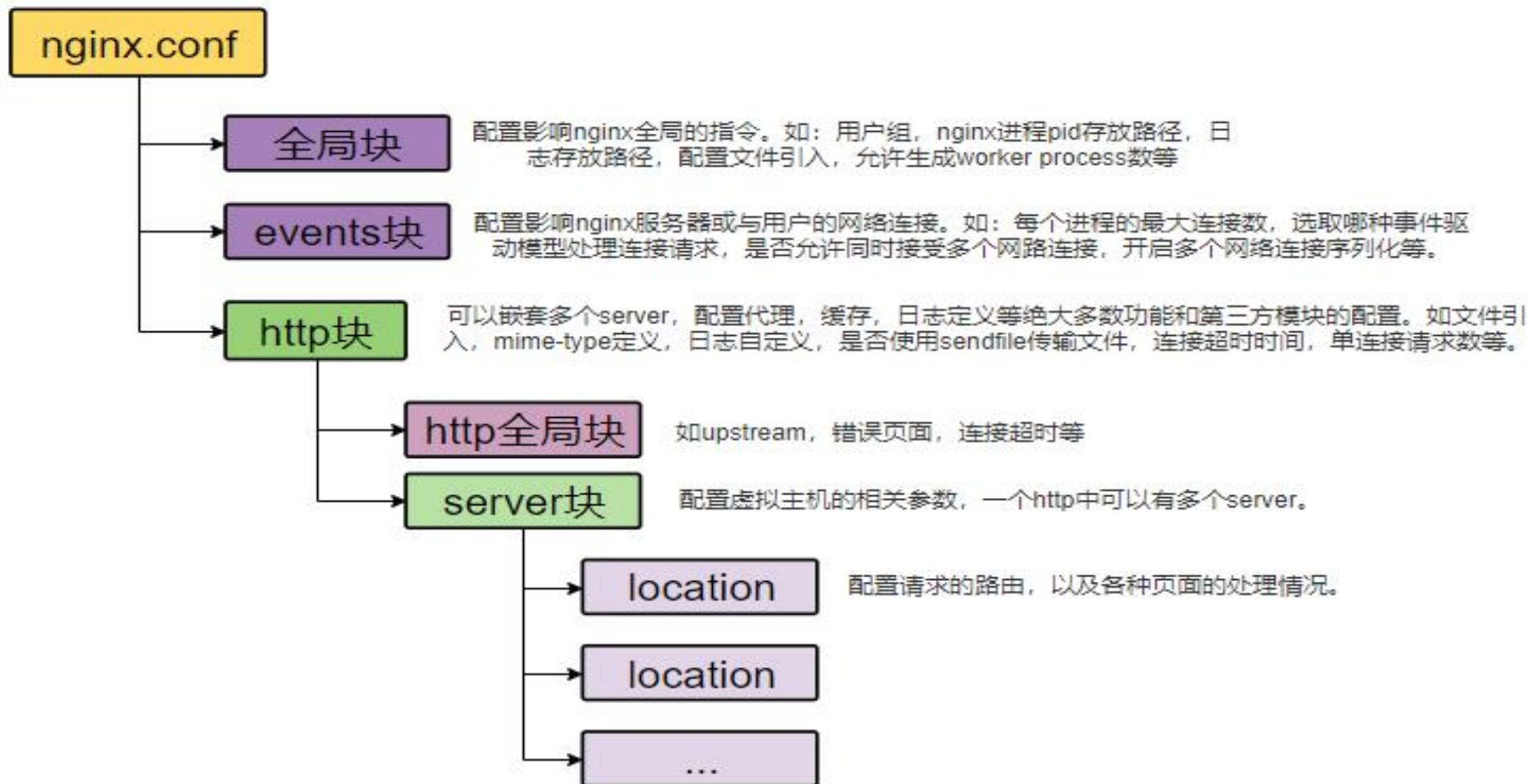


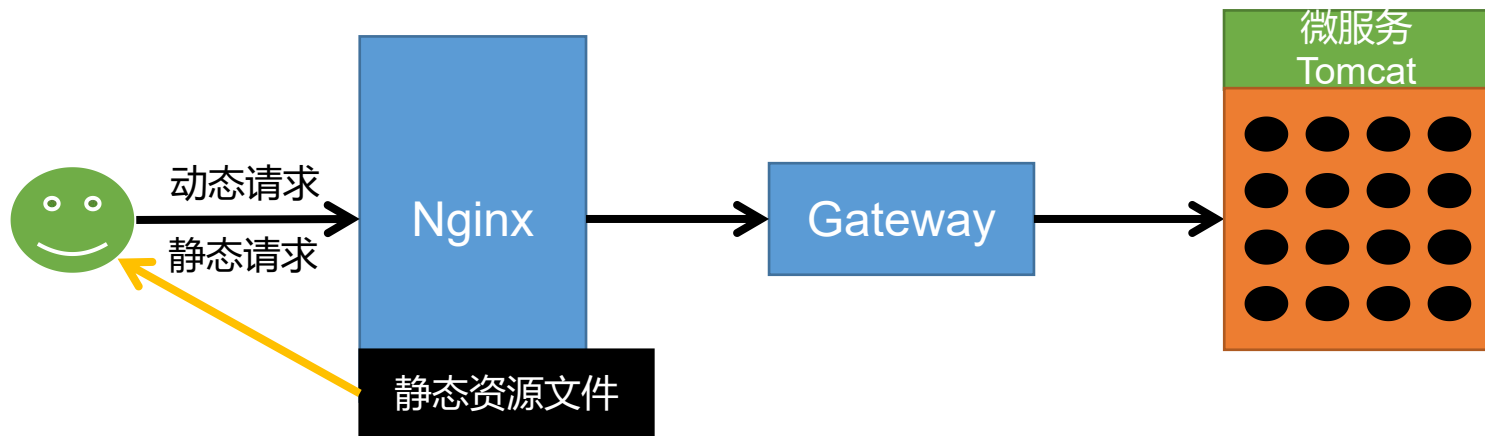
正向代理：如科学上网，隐藏客户端信息



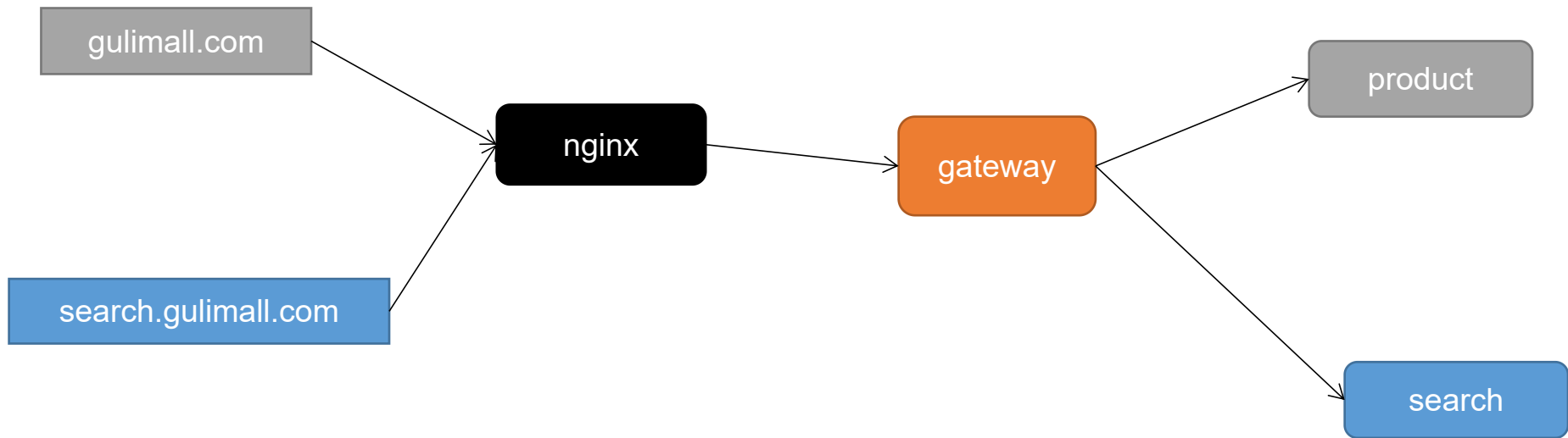
反向代理：屏蔽内网服务器信息，负载均衡访问

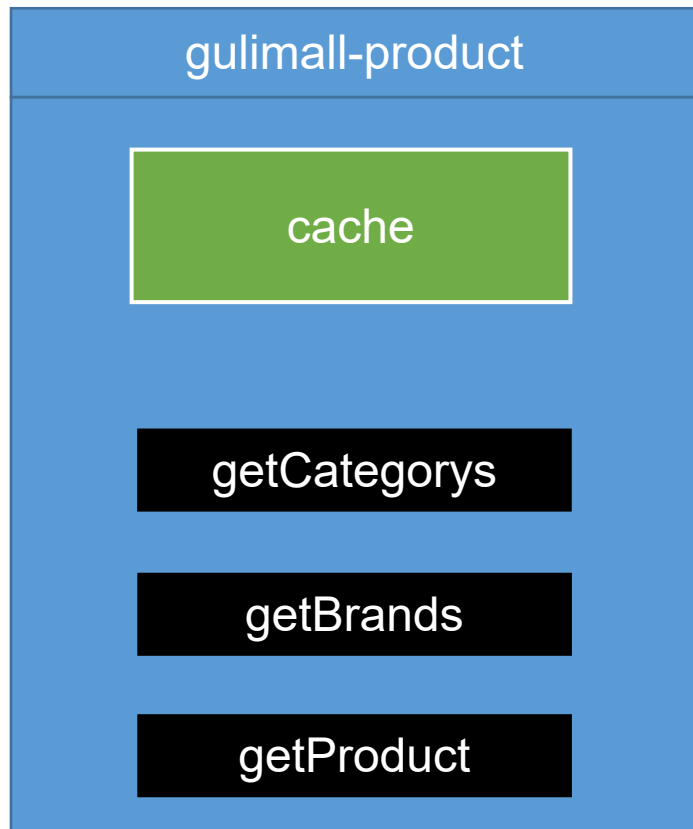


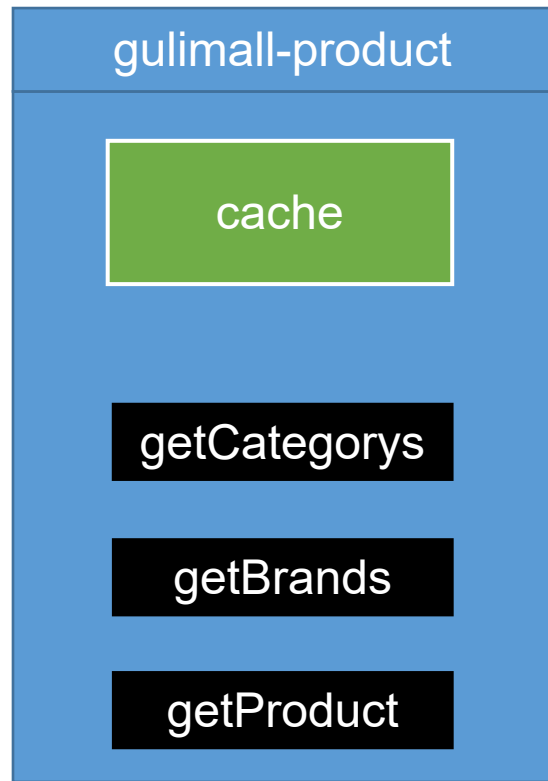
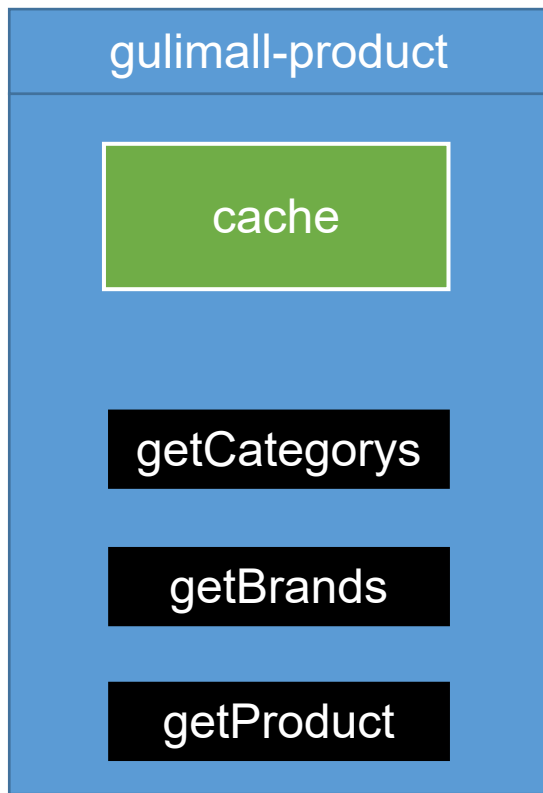
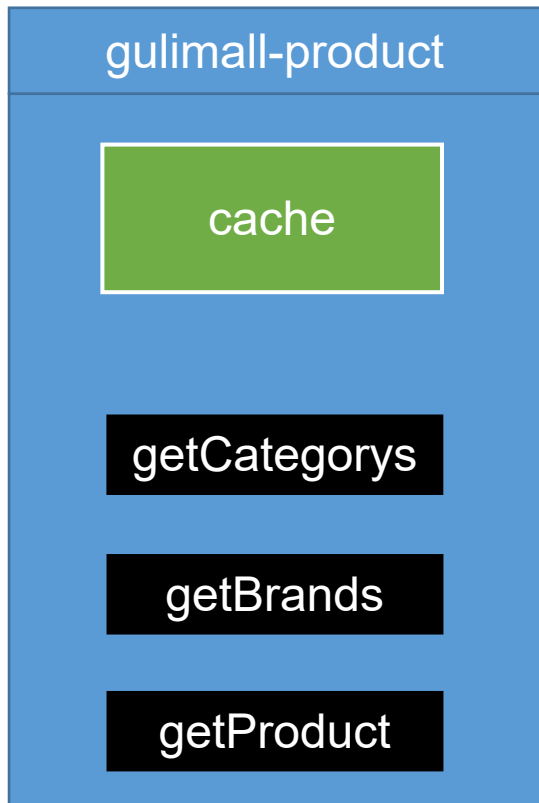


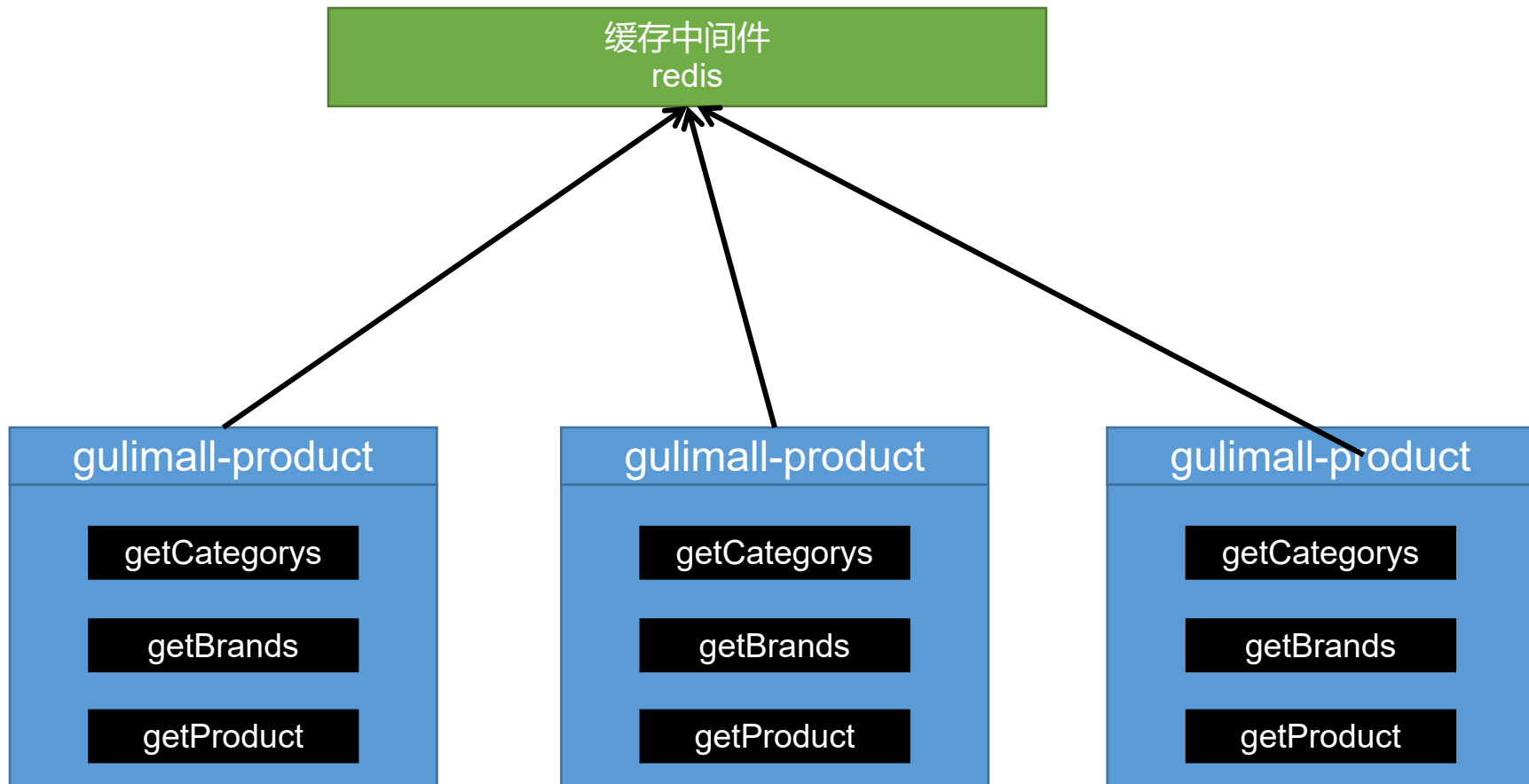


- 1、以后将所有项目的静态资源都应该放在nginx里面
- 2、规则：/static/**所有请求都由nginx直接返回











缓存穿透：

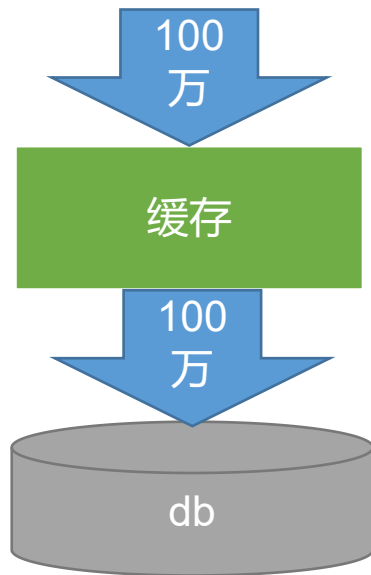
指查询一个一定不存在的数据，由于缓存是不命中，将去查询数据库，但是数据库也无此记录，我们没有将这次查询的null写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义

风险：

利用不存在的数据进行攻击，数据库瞬时压力增大，最终导致崩溃

解决：

null结果缓存，并加入短暂过期时间



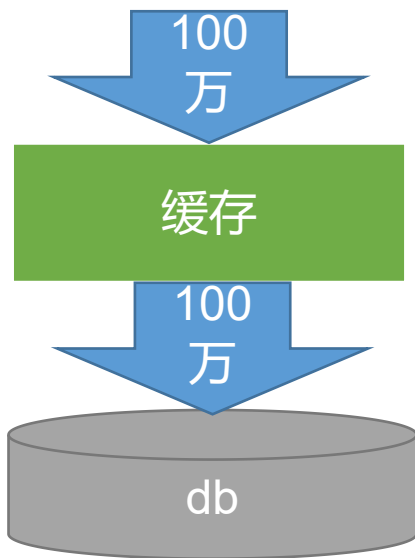


缓存雪崩:

缓存雪崩是指在我们设置缓存时key采用了相同的过期时间, 导致缓存在某一时刻同时失效, 请求全部转发到DB, DB瞬时压力过重雪崩。

解决:

原有的失效时间基础上增加一个随机值, 比如1-5分钟随机, 这样每一个缓存的过期时间的重复率就会降低, 就很难引发集体失效的事件。





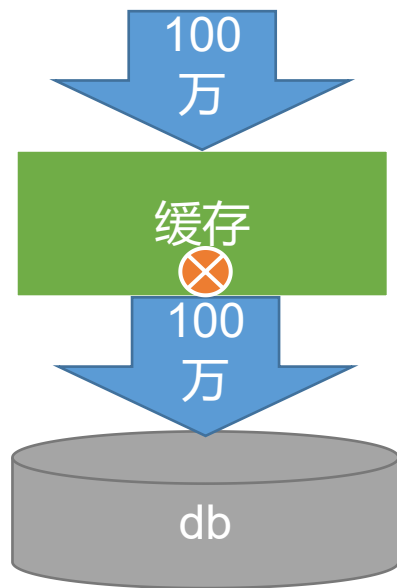
缓存穿透:

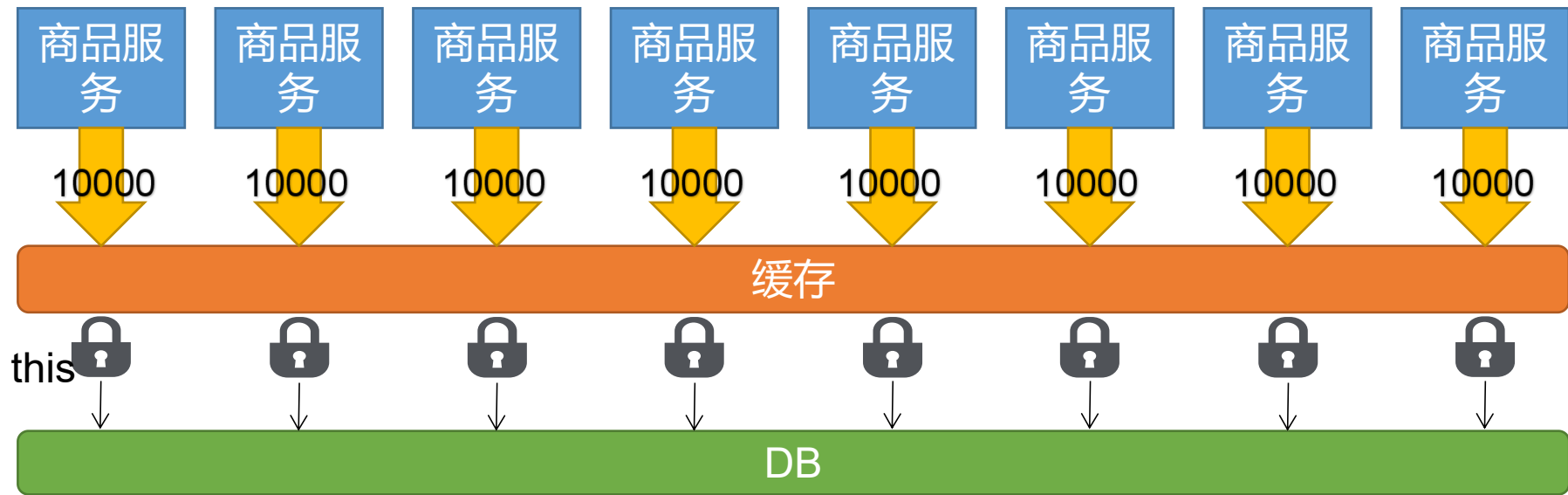
- 对于一些设置了过期时间的key, 如果这些key可能会在某些时间点被超高并发地访问, 是一种非常“热点”的数据。
- 如果这个key在大量请求同时进来前正好失效, 那么所有对这个key的数据查询都落到db, 我们称为缓存击穿。

解决:

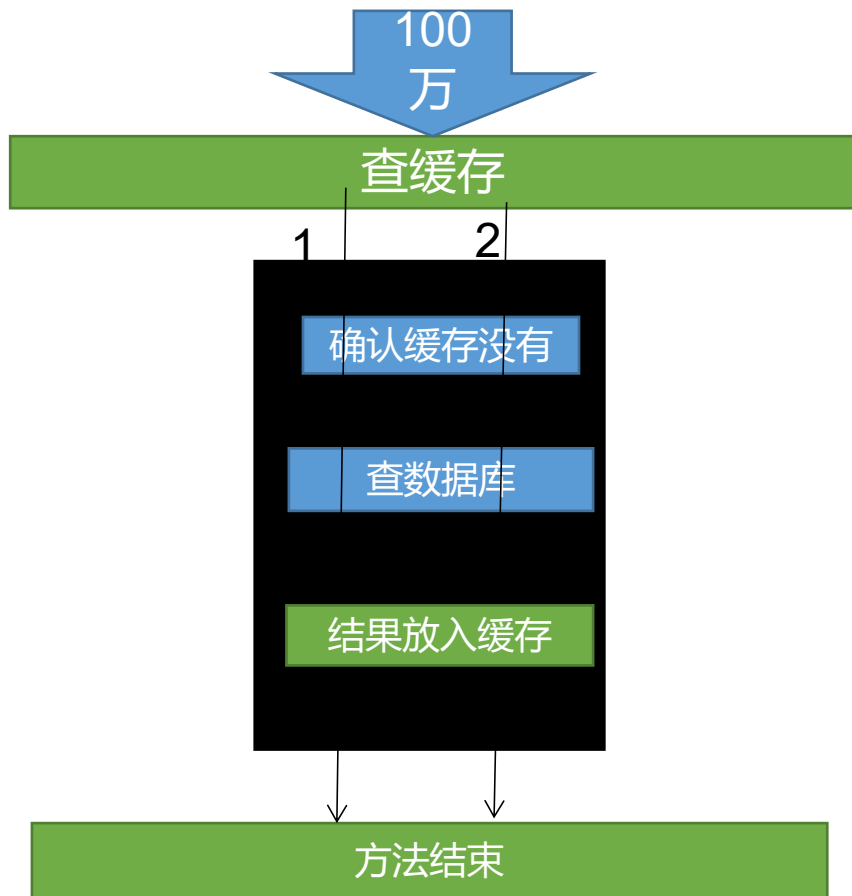
加锁

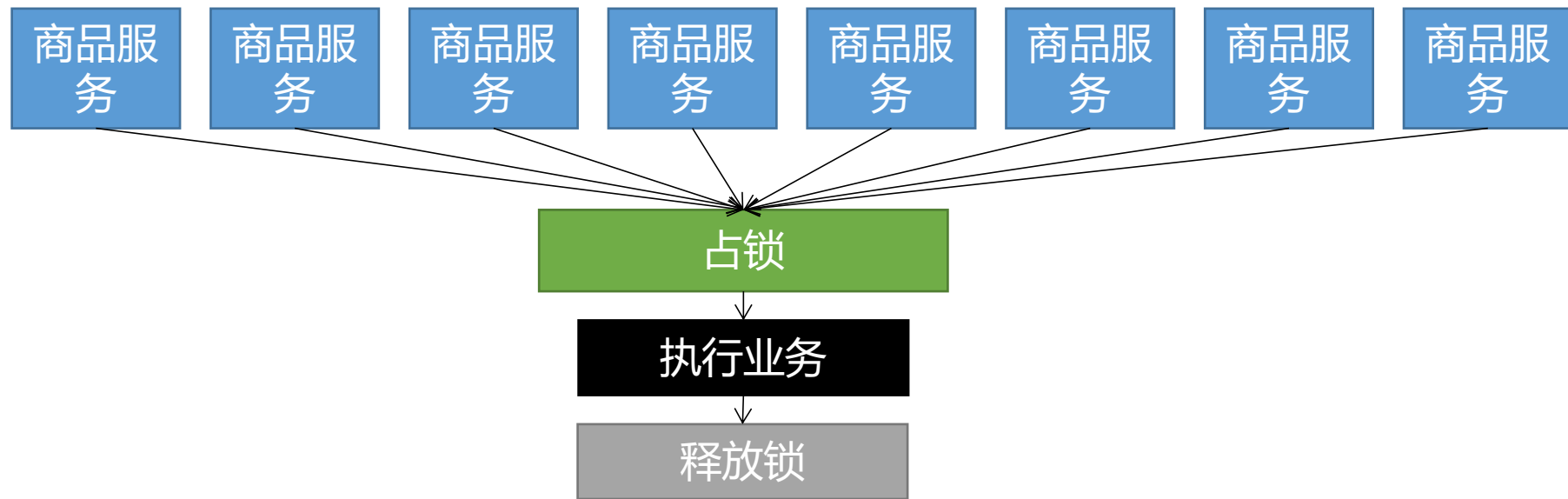
大量并发只让一个去查, 其他人等待, 查到以后释放锁, 其他人获取到锁, 先查缓存, 就会有数据, 不用去db



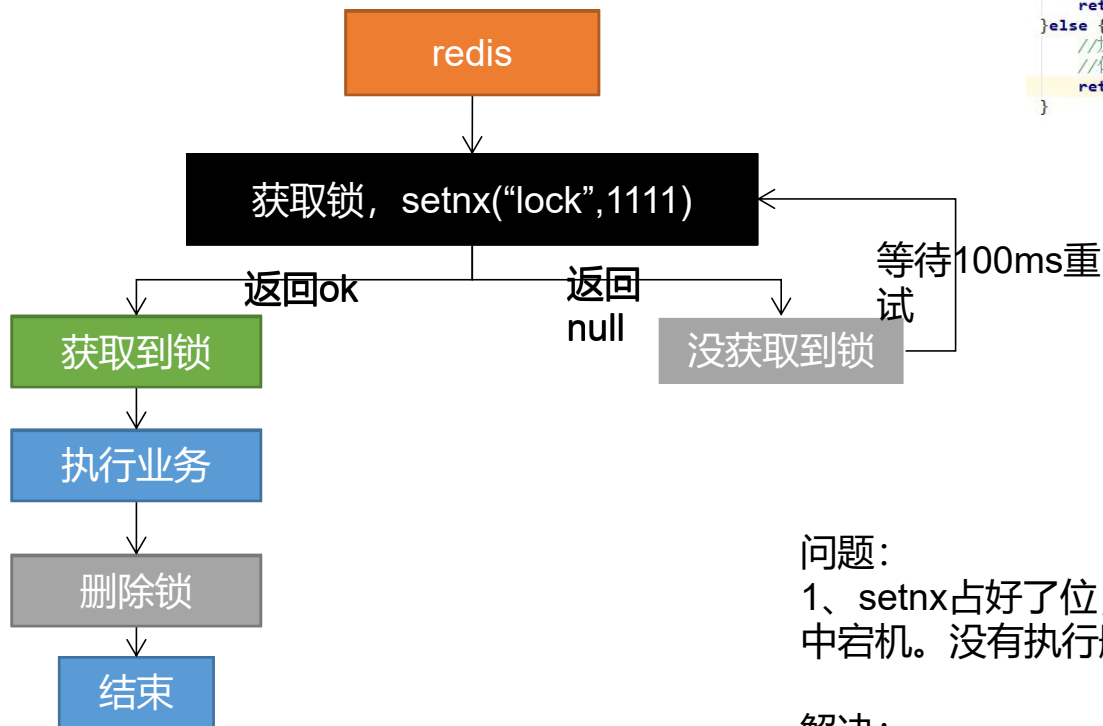


本地锁，只能锁住当前进程，所以我们需要分布式锁





我们可以同时去一个地方“占坑”，如果占到，就执行逻辑。否则就必须等待，直到释放锁。
“占坑”可以去redis，可以去数据库，可以去任何大家都能访问的地方。
等待可以**自旋**的方式。



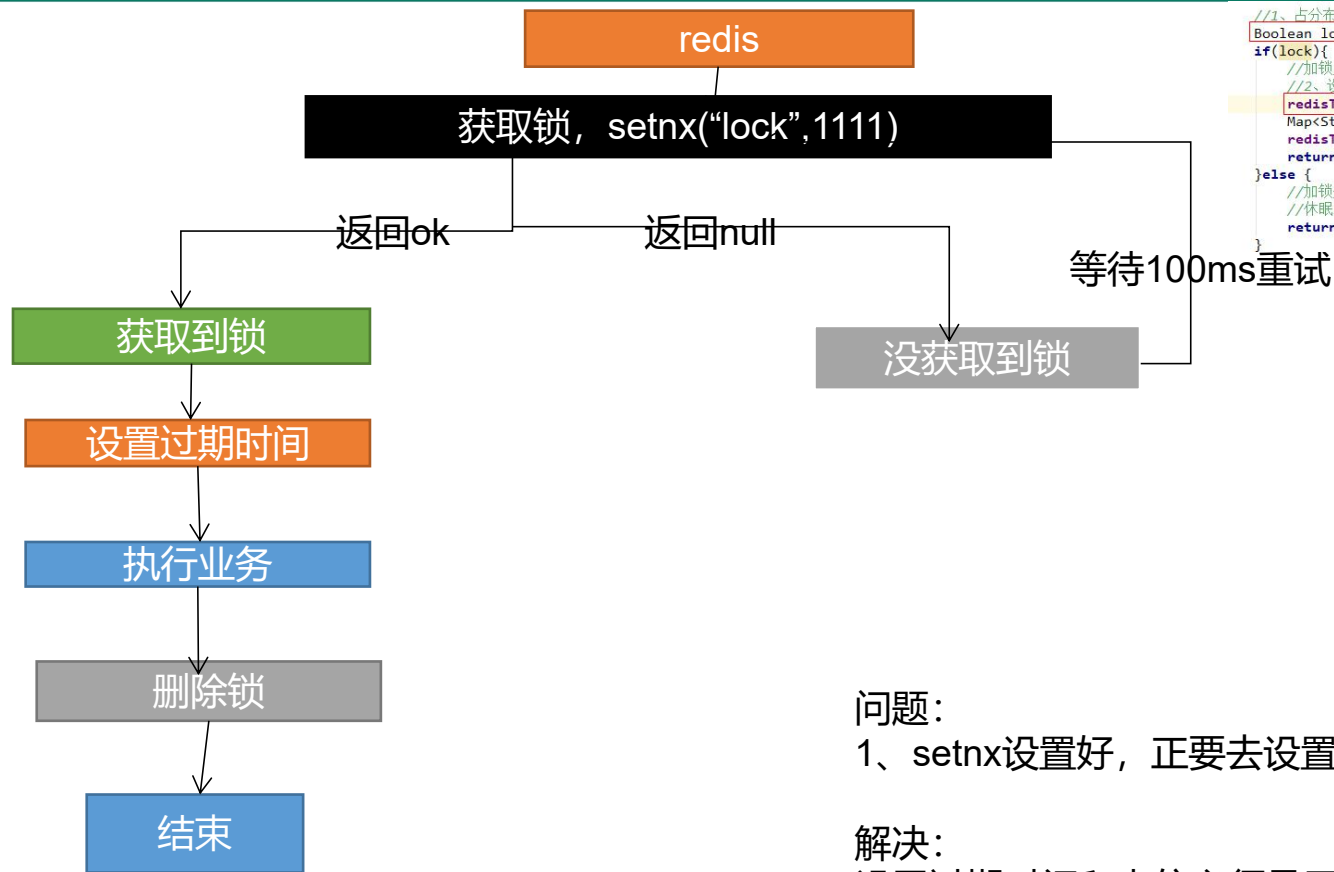
```
//1、占分布式锁。去redis占坑
Boolean lock = redisTemplate.opsForValue().setIfAbsent("lock", "111");
if(lock){
    //加锁成功... 执行业务
    Map<String, List<Catalog2Vo>> dataFromDb = getDataFromDb();
    redisTemplate.delete( key: "lock"); //删除锁
    return dataFromDb;
}else {
    //加锁失败...重试。synchronized ()
    //休眠100ms重试
    return getCatalogJsonFromDbWithRedisLock(); //自旋的方式
}
```

问题：

1、`setnx`占好了位，业务代码异常或者程序在页面过程中宕机。没有执行删除锁逻辑，这就造成了**死锁**

解决：

设置锁的自动过期，即使没有删除，会自动删除



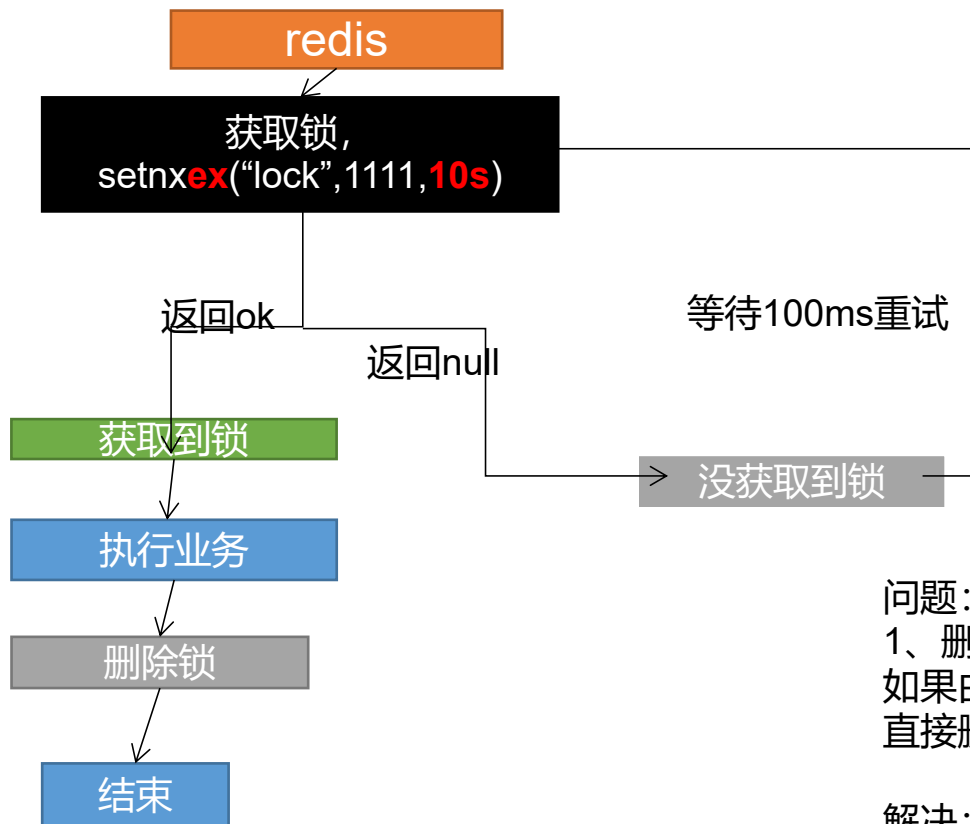
```
//1、占分布式锁。去redis占坑
Boolean lock = redisTemplate.opsForValue().setIfAbsent("lock", "111");
if(lock){
    //加锁成功... 执行业务
    //2、设置过期时间
    redisTemplate.expire(key: "lock", timeout: 30, TimeUnit.SECONDS);
    Map<String, List<Catalog2Vo>> dataFromDb = getDataFromDb();
    redisTemplate.delete(key: "lock"); //删除锁
    return dataFromDb;
} else {
    //加锁失败... 重试。synchronized ()
    //休眠100ms重试
    return getCatalogJsonFromDbWithRedisLock(); //自旋的方式
}
```

问题：

1、setnx设置好，正要去设置过期时间，宕机。又死锁了。

解决：

设置过期时间和占位必须是原子的。redis支持使用setnx ex 命令



```
//1、占分布式锁。去redis占坑
Boolean lock = redisTemplate.opsForValue().setIfAbsent( key: "lock", value: "1111", timeout: 300, TimeUnit.SECONDS);
if(lock){
    //加锁成功... 执行业务
    //2、设置过期时间, 必须和加锁是同步的, 原子的
    //redisTemplate.expire("lock", 30, TimeUnit.SECONDS);
    Map<String, List<Catalog2Vo>> dataFromDb = getDataFromDb();
    redisTemplate.delete( key: "lock"); //删除锁
    return dataFromDb;
}else {
    //加锁失败...重试。synchronized ()
    //休眠100ms重试
    return getCatalogJsonFromDbWithRedisLock(); //自旋的方式
}
```

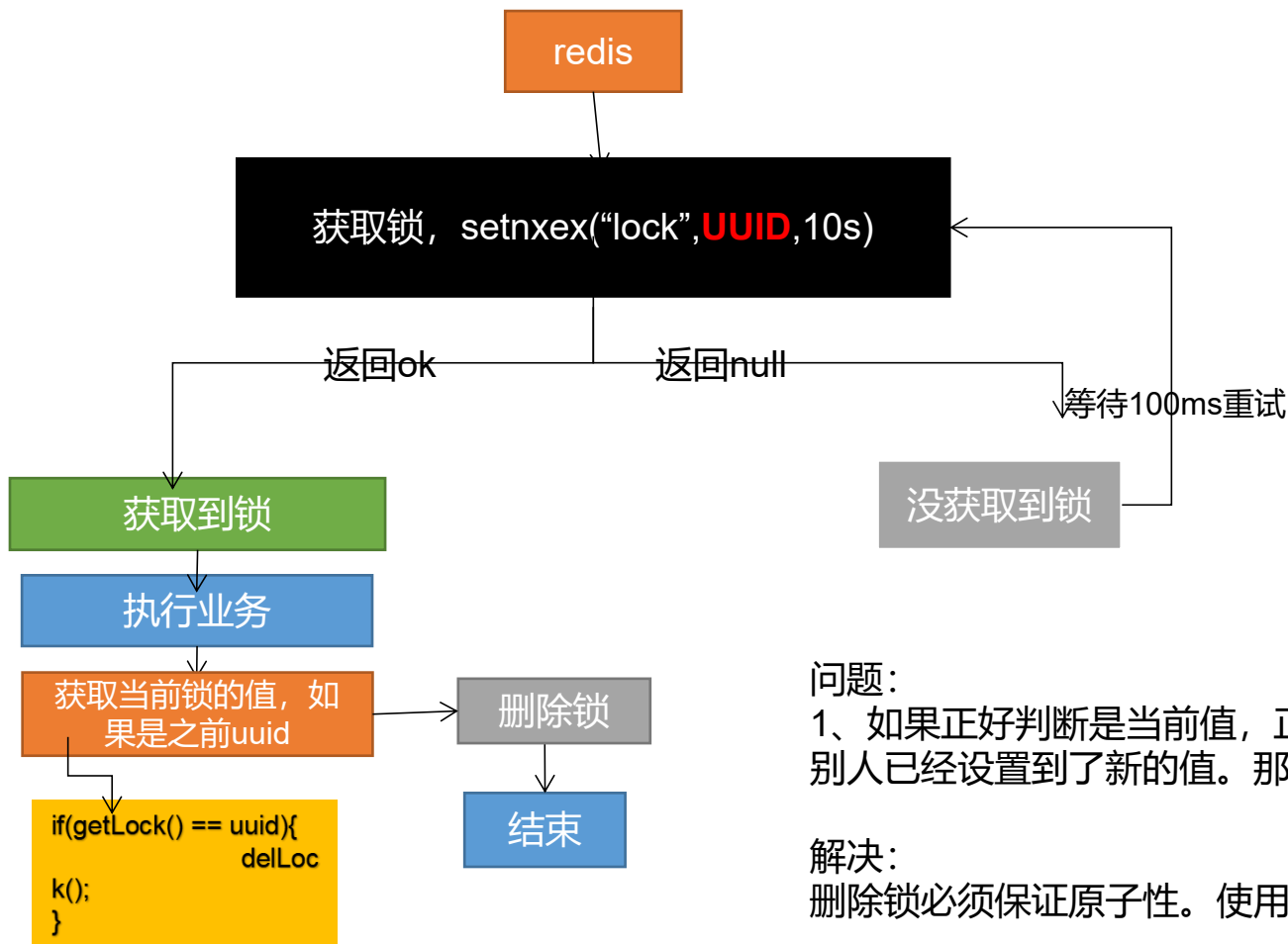
问题:

1、删除锁直接删除???

如果由于业务时间很长, 锁自己过期了, 我们直接删除, 有可能把别人正在持有的锁删除了。

解决:

占锁的时候, 值指定为uuid, 每个人匹配是自己的锁才删除。

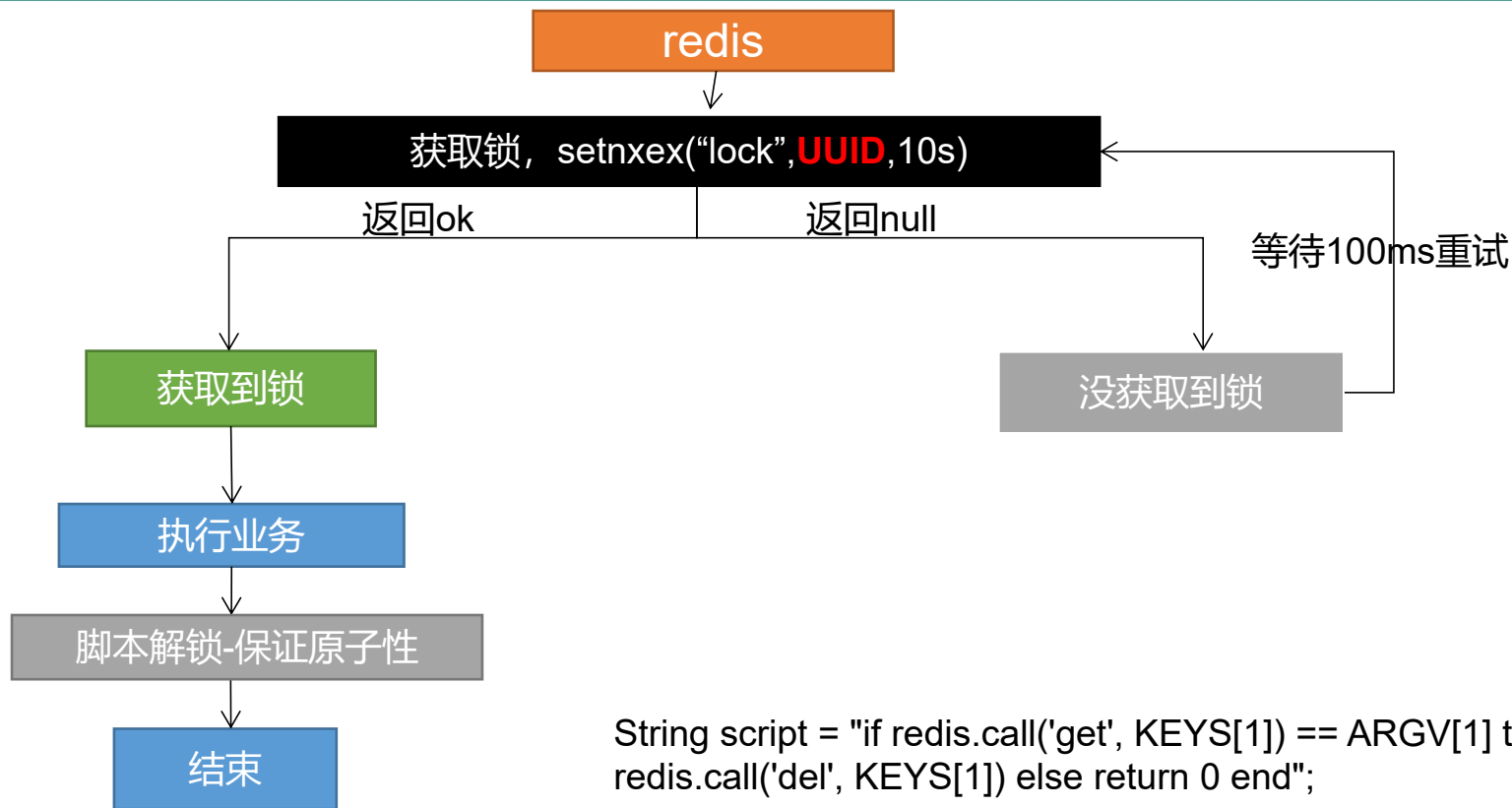


问题:

1、如果正好判断是当前值，正要删除锁的时候，锁已经过期，别人已经设置到了新的值。那么我们删除的是别人的锁

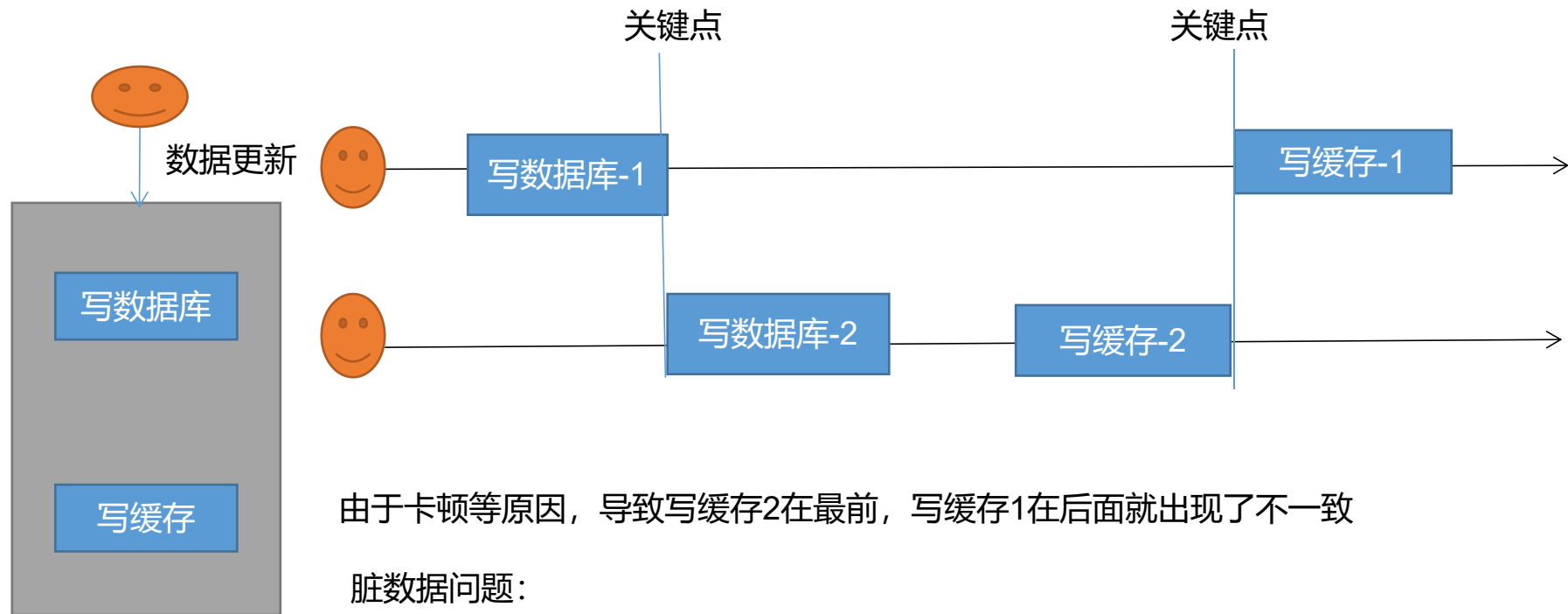
解决:

删除锁必须保证原子性。使用redis+Lua脚本完成



```
String script = "if redis.call('get', KEYS[1]) == ARGV[1] then return\nredis.call('del', KEYS[1]) else return 0 end";
```

保证加锁【占位+过期时间】和删除锁【判断+删除】的原子性。
更难的事情，锁的自动续期

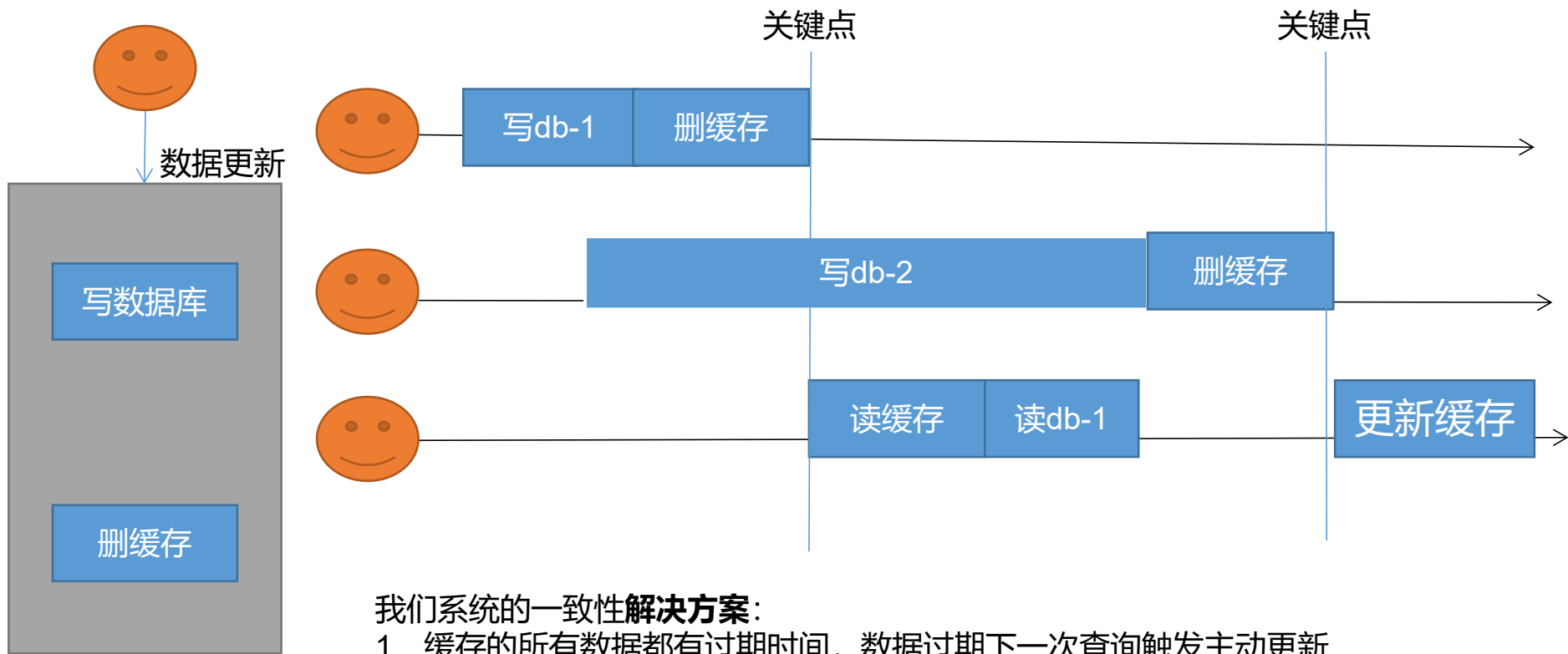


由于卡顿等原因，导致写缓存2在最前，写缓存1在后面就出现了不一致

脏数据问题：

这是暂时性的脏数据问题，但是在数据稳定，缓存过期以后，又能得到最新的正确数据

读到的最新数据有延迟：最终一致性



我们系统的一致性**解决方案**:

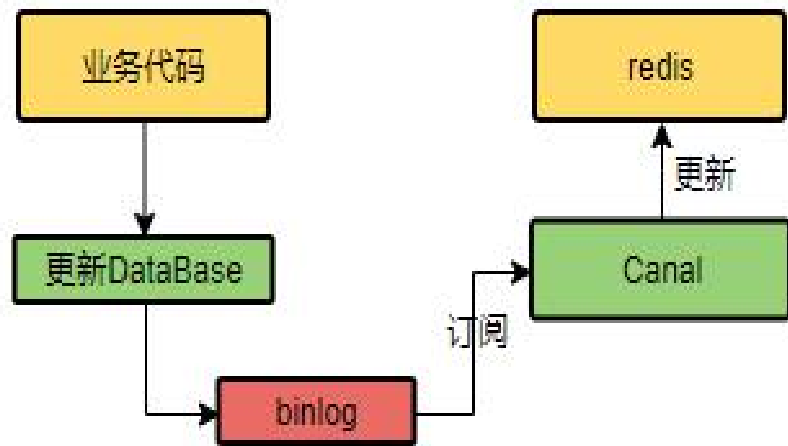
- 1、缓存的所有数据都有过期时间，数据过期下一次查询触发主动更新
- 2、读写数据的时候，加上分布式的**读写锁**。
经常写，经常读



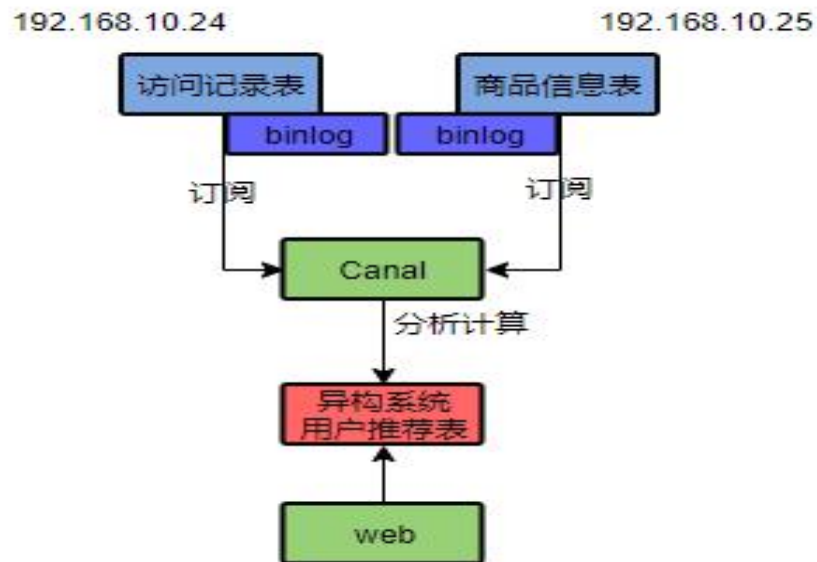
- 无论是双写模式还是失效模式，都会导致缓存的不一致问题。即多个实例同时更新会出事。怎么办？
 - 1、如果是用户纬度数据（订单数据、用户数据），这种并发几率非常小，不用考虑这个问题，缓存数据加上过期时间，每隔一段时间触发读的主动更新即可
 - 2、如果是菜单，商品介绍等基础数据，也可以去使用canal订阅binlog的方式。
 - 3、缓存数据+过期时间也足够解决大部分业务对于缓存的要求。
 - 4、通过加锁保证并发读写，写写的时候按顺序排好队。读读无所谓。所以适合使用读写锁。（业务不关心脏数据，允许临时脏数据可忽略）；
- 总结：
 - 我们能放入缓存的数据本就不应该是实时性、一致性要求超高的。所以缓存数据的时候加上过期时间，保证每天拿到当前最新数据即可。
 - 我们不应该过度设计，增加系统的复杂性
 - 遇到实时性、一致性要求高的数据，就应该查数据库，即使慢点。



使用Canal更新缓存



使用Canal解决数据异构

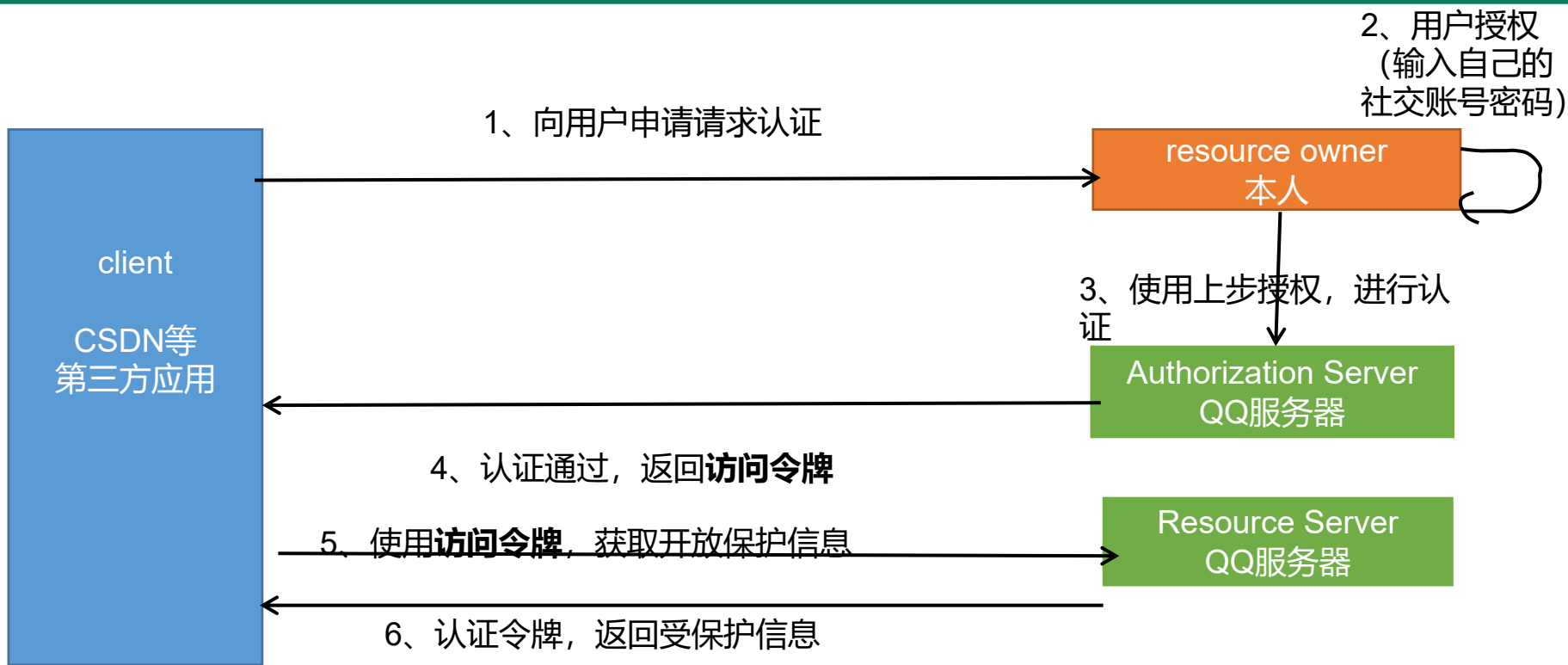




- 1、全文检索: skuTitle-》 keyword
 - 2、排序: saleCount (销量)、hotScore (热度分)、skuPrice (价格)
 - 3、过滤: hasStock、skuPrice区间、brandId、catalog3Id、attrs
 - 4、聚合: attrs
-
- 完整查询参数
 - keyword=小米
&sort=saleCount_desc/asc&hasStock=0/1&skuPrice=400_1900&brandId=1&catalog3Id=1&attrs=1_3G:4G:5G&attrs=2_骁龙845&attrs=4_高清屏



- MD5
 - Message Digest algorithm 5, 信息摘要算法
 - 压缩性: 任意长度的数据, 算出的MD5值长度都是固定的。
 - 容易计算: 从原数据计算出MD5值很容易。
 - 抗修改性: 对原数据进行任何改动, 哪怕只修改1个字节, 所得到的MD5值都有很大区别。
 - 强抗碰撞: 想找到两个不同的数据, 使它们具有相同的MD5值, 是非常困难的。
 - 不可逆
- 加盐:
 - 通过生成随机数与MD5生成字符串进行组合
 - 数据库同时存储MD5值与salt值。验证正确性时使用salt进行MD5即可



1、使用Code换取AccessToken, Code只能用一次

2、同一个用户的accessToken一段时间是不会变化的, 即使多次获取



Session共享问题-session原理

Name	Value	Domain
JSESSIONID	BBF4769EB6DD96EB9BC8A133273662B0	auth.gulimall.com

1、第一次访问服务器（进行登录）

2、登录成功
用户保存到
session中

2、命令浏览器保存一个jsessionId=123的cookie

3、以后访问会带上cookie。jsessionid=123

服务器

4、浏览器关闭，清除会话cookie

5、下次访问，没有jsessionid，再创建一个，进入步骤1

123
user=v,k=v

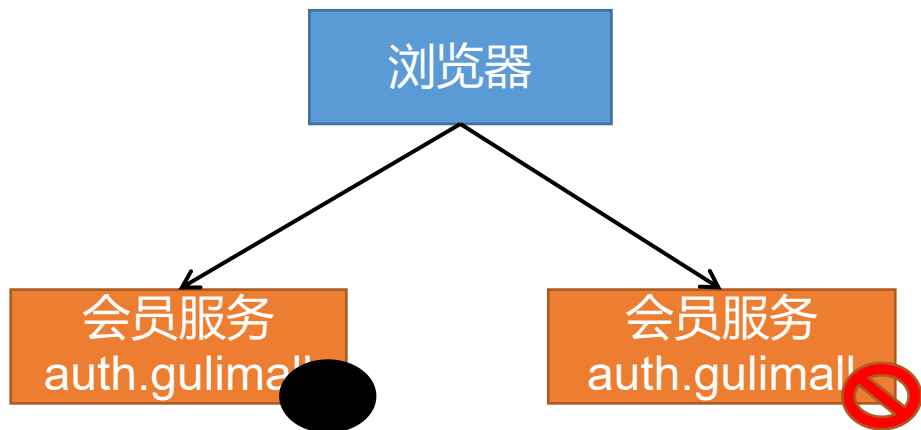
234
k=v,k=v

567

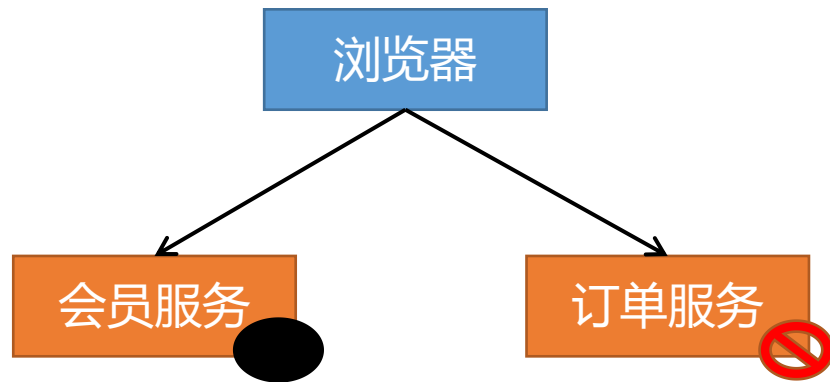
sessionManager

问题：1、不能跨不同域名共享

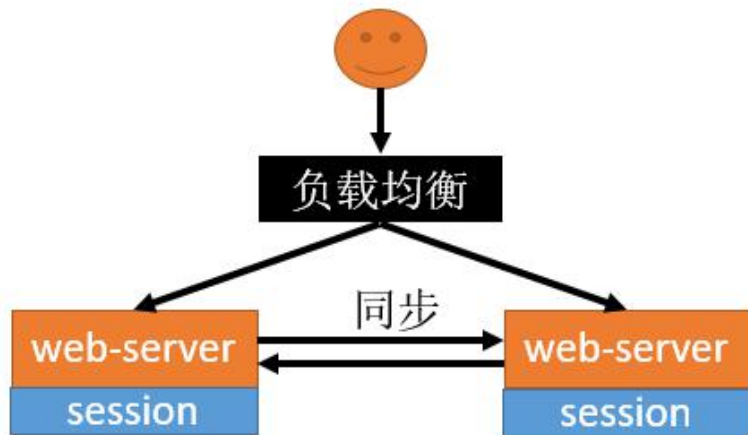
浏览器



同一个服务，复制多份，session不同步问题

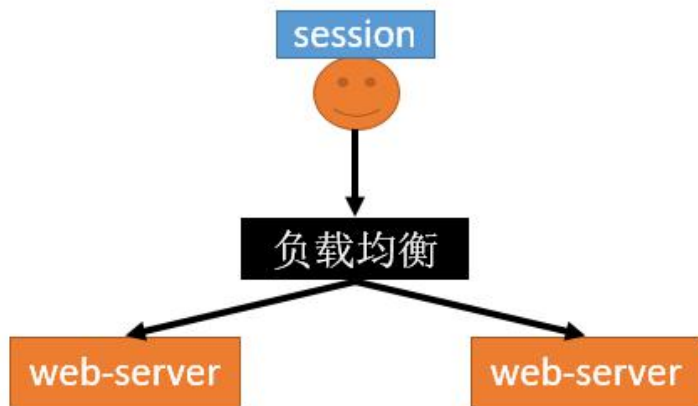


不同服务，session不能共享问题



session复制（同步）方案

- 优点
 - web-server (Tomcat) 原生支持，只需要修改配置文件
- 缺点
 - session同步需要数据传输，占用大量网络带宽，降低了服务器群的业务处理能力
 - 任意一台web-server保存的数据都是所有web-server的session总和，受到内存限制无法水平扩展更多的web-server
 - 大型分布式集群情况下，由于所有web-server都全量保存数据，所以此方案不可取。

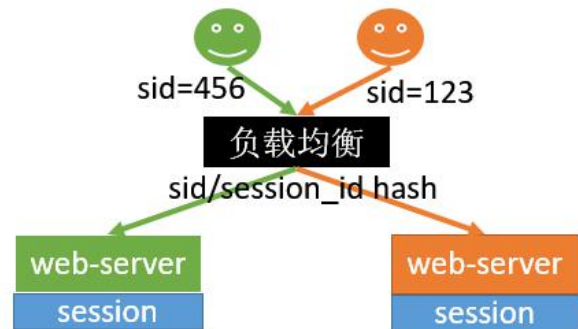


session存储在客户端cookie中

- 优点
 - 服务器不需存储session，用户保存自己的session信息到cookie中。节省服务端资源
- 缺点
 - 都是缺点，这只是一种思路。
 - 具体如下：
 - 每次http请求，携带用户在cookie中的完整信息，浪费网络带宽
 - session数据放在cookie中，cookie有长度限制4K，不能保存大量信息
 - session数据放在cookie中，存在泄漏、篡改、窃取等安全隐患
- 这种方式不会使用。

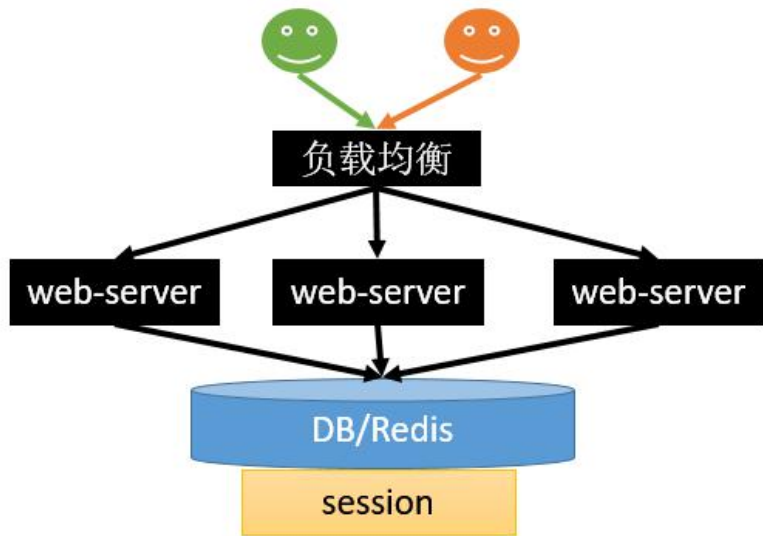


四层代理ip_hash方案



七层代理业务字段hash方案

- 优点:
 - 只需要改nginx配置, 不需要修改应用代码
 - 负载均衡, 只要hash属性的值分布是均匀的, 多台web-server的负载是均衡的
 - 可以支持web-server水平扩展 (session同步法是不行的, 受内存限制)
- 缺点
 - session还是存在web-server中的, 所以web-server重启可能导致部分session丢失, 影响业务, 如部分用户需要重新登录
 - 如果web-server水平扩展, rehash后session重新分布, 也会有一部分用户路由不到正确的session
- 但是以上缺点问题也不是很大, 因为session本来都是有有效期的。所以这两种反向代理的方式可以使用



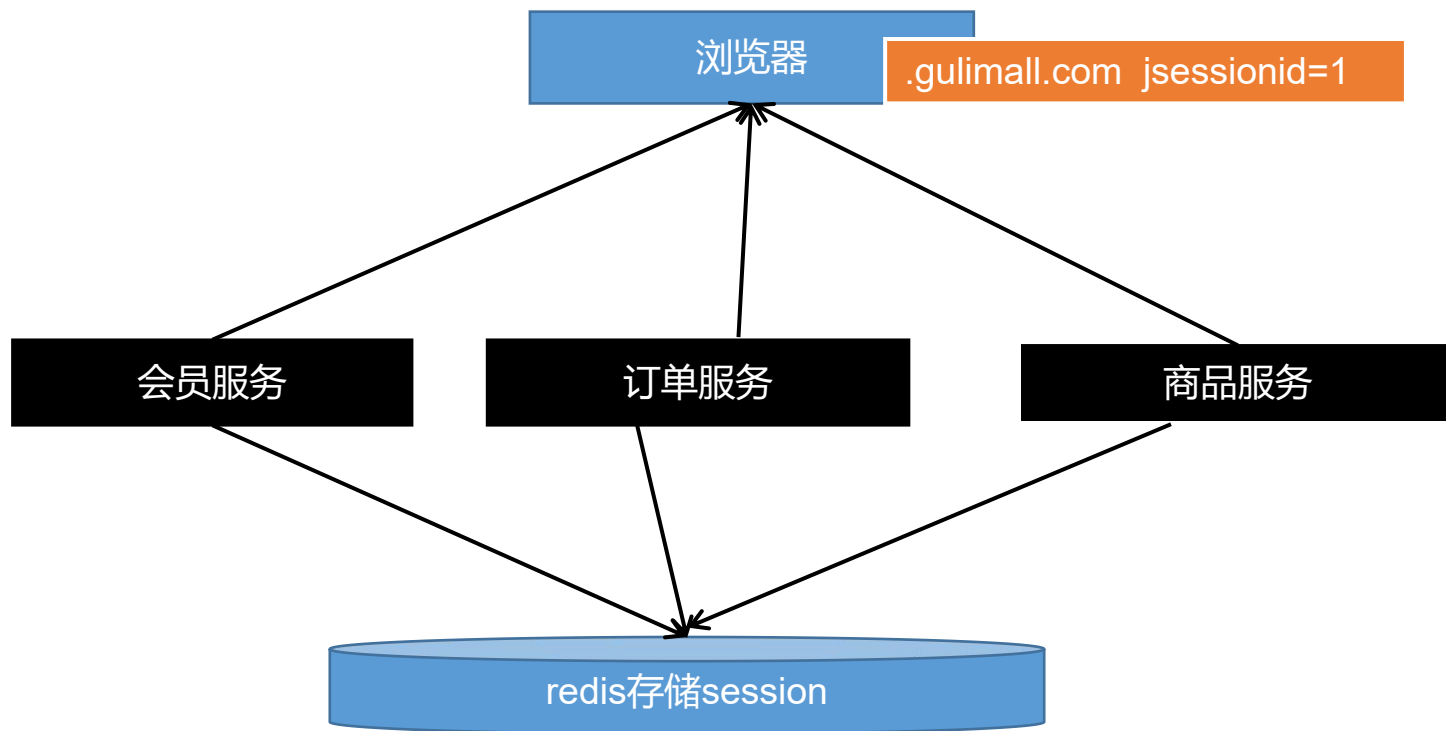
后端统一存储session

- 优点:
 - 没有安全隐患
 - 可以水平扩展，数据库/缓存水平切分即可
 - web-server重启或者扩容都不会有session丢失
- 不足
 - 增加了一次网络调用，并且需要修改应用代码；如将所有的getSession方法替换为从Redis查数据的方式。redis获取数据比内存慢很多
 - 上面缺点可以用SpringSession完美解决



Session共享问题解决-不同服务，子域session共享

jsessionid这个cookie默认是当前系统域名的。当我们分拆服务，不同域名部署的时候，我们可以使用如下解决方案；





SessionRepositoryFilter

@Override

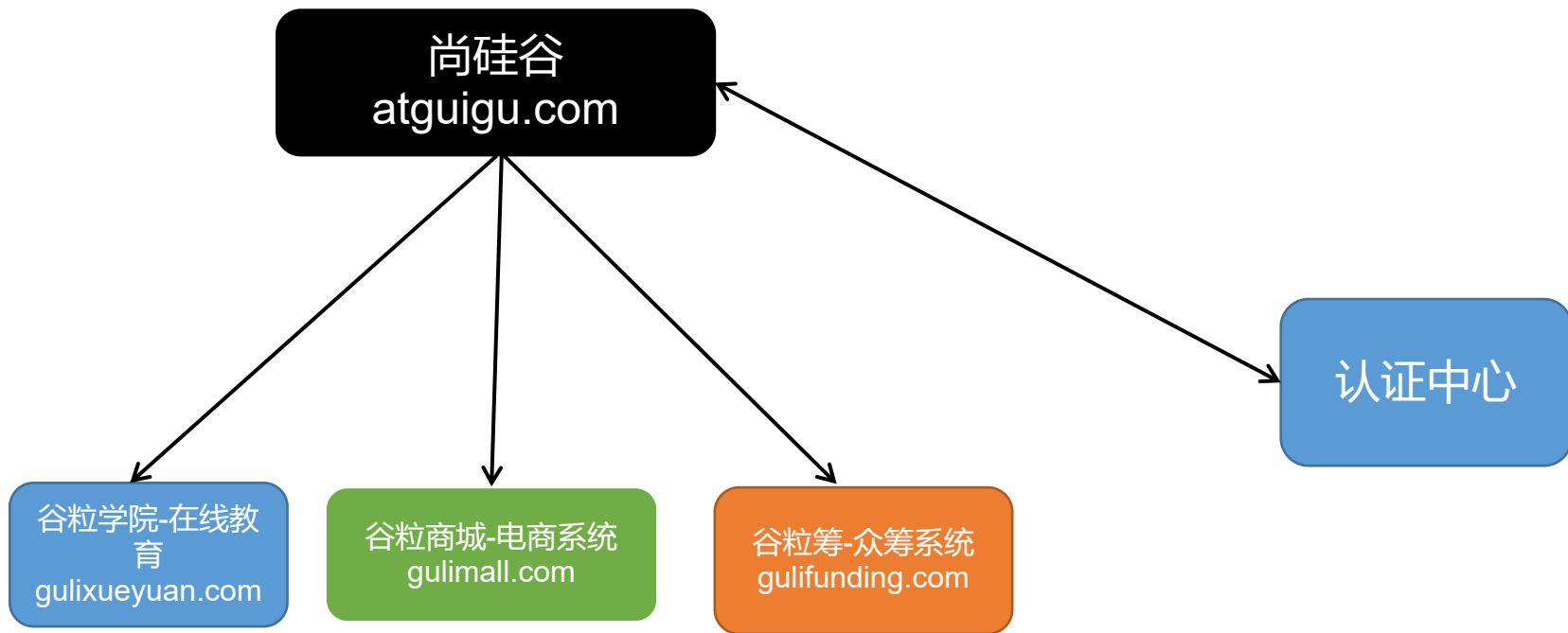
```
protected void doFilterInternal(HttpServletRequest request,
    HttpServletResponse response, FilterChain filterChain)
    throws ServletException, IOException {
    request.setAttribute(SESSION_REPOSITORY_ATTR, this.sessionRepository); 包装原始请求对象。
```

```
    SessionRepositoryRequestWrapper wrappedRequest = new SessionRepositoryRequestWrapper(
        request, response, this.servletContext);
    SessionRepositoryResponseWrapper wrappedResponse = new SessionRepositoryResponseWrapper(
        wrappedRequest, response);
```

```
    try {
        filterChain.doFilter(wrappedRequest, wrappedResponse);
    }
    finally {
        wrappedRequest.commitSession();
    }
}
```

包装后的对象应用到了我们后面的整个执行链

HttpSession session1 = request.getSession();





- /xxl-sso-server 登录服务器 8080 ssoserver.com
- /xxl-sso-web-sample-springboot 项目1 8081 client1.com
- /xxl-sso-web-sample-springboot 项目2 8082 client2.com

```
127.0.0.1    ssoserver.com  
127.0.0.1    client1.com  
127.0.0.1    client2.com
```

核心：三个系统即使域名不一样，想办法给三个系统同步同一个用户的票据；

- 1)、中央认证服务器；ssoserver.com
- 2)、其他系统，想要登录去ssoserver.com登录，登录成功跳转回来
- 3)、只要有一个登录，其他都不用登录
- 4)、全系统统一一个sso-sessionid；所有系统可能域名都不相同



购物车数据结构

HASH: gulimall:cart:6

Size: 4 TTL: -1 Re

row	key	value
1	2	{"check":true,"count":5,"image":"https://gulimall-he..."}
2	4	{"check":true,"count":3,"image":"https://gulimall-he..."}
3	7	{"check":true,"count":27,"image":"https://gulimall-h..."}
4	5	{"check":true,"count":6,"image":"https://gulimall-he..."}

cart: 1

1

{skuld:1,title:"华为",price:2999}

7

{skuld:7,title:"小米",price:2999}

cart: 2

11

{skuld:11,title:"华为P30",price:2999}

23

{skuld:23,title:"小米9",price:2999}

Map<String k1,Map<String k2,CartItemInfo>>

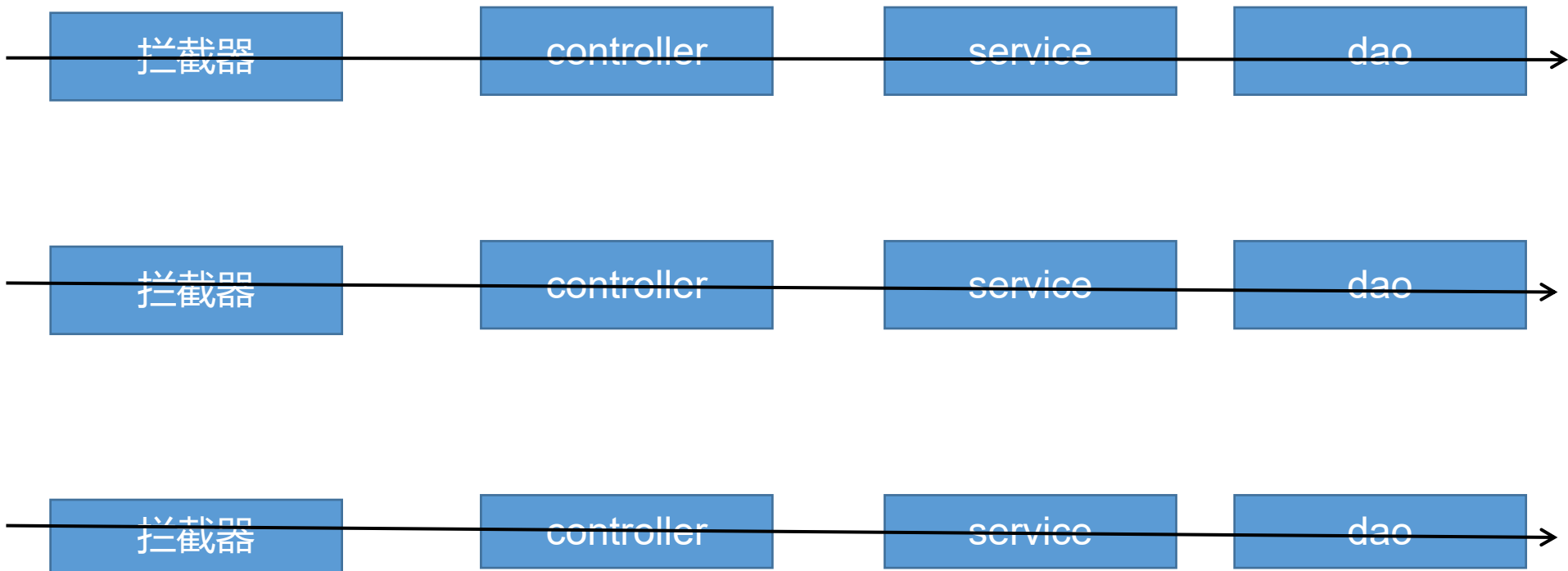
k1: 标识每一个用户的购物车

k2: 购物项的商品id

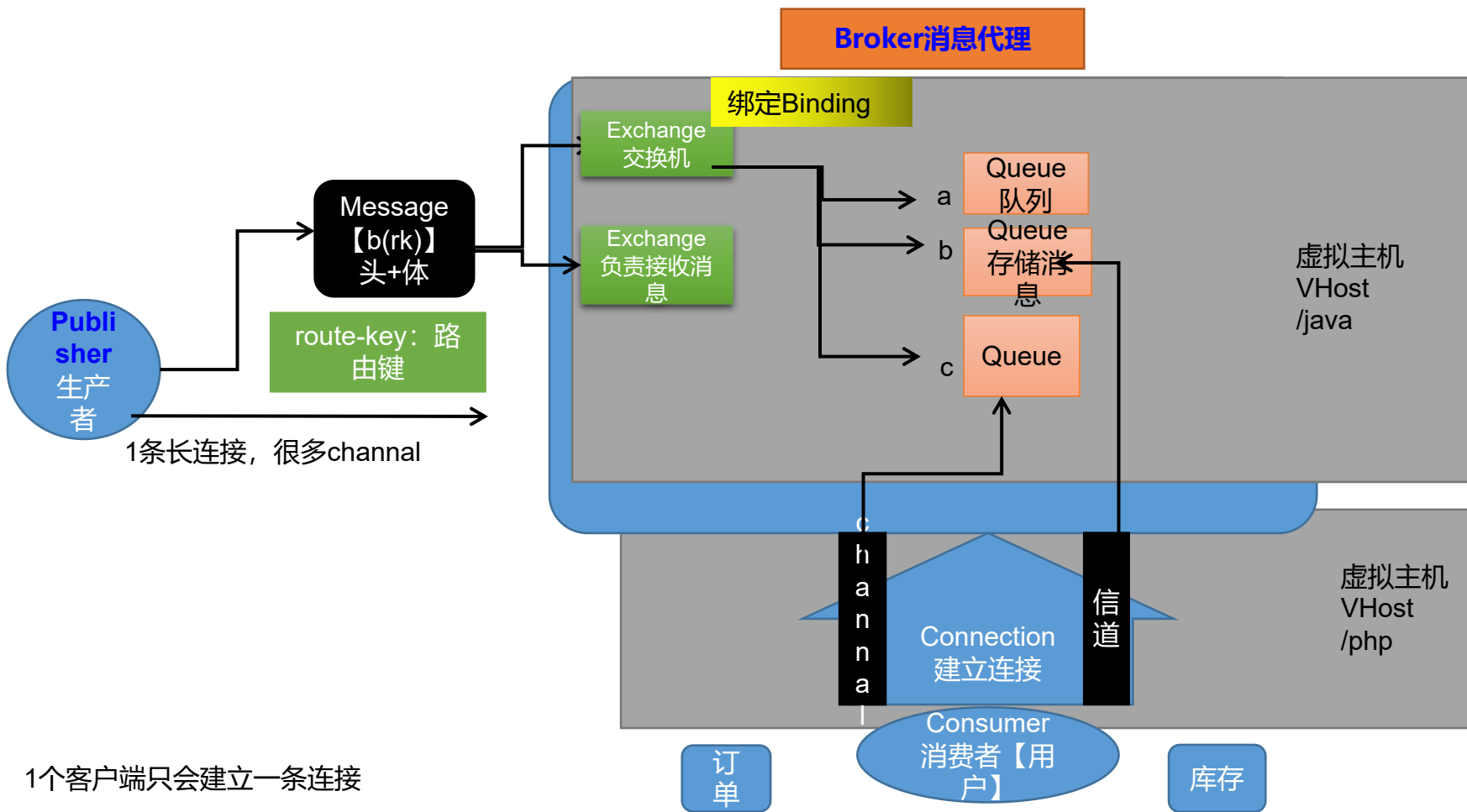
在redis中

key:用户标识

value:Hash (k: 商品id, v: 购物项详情)

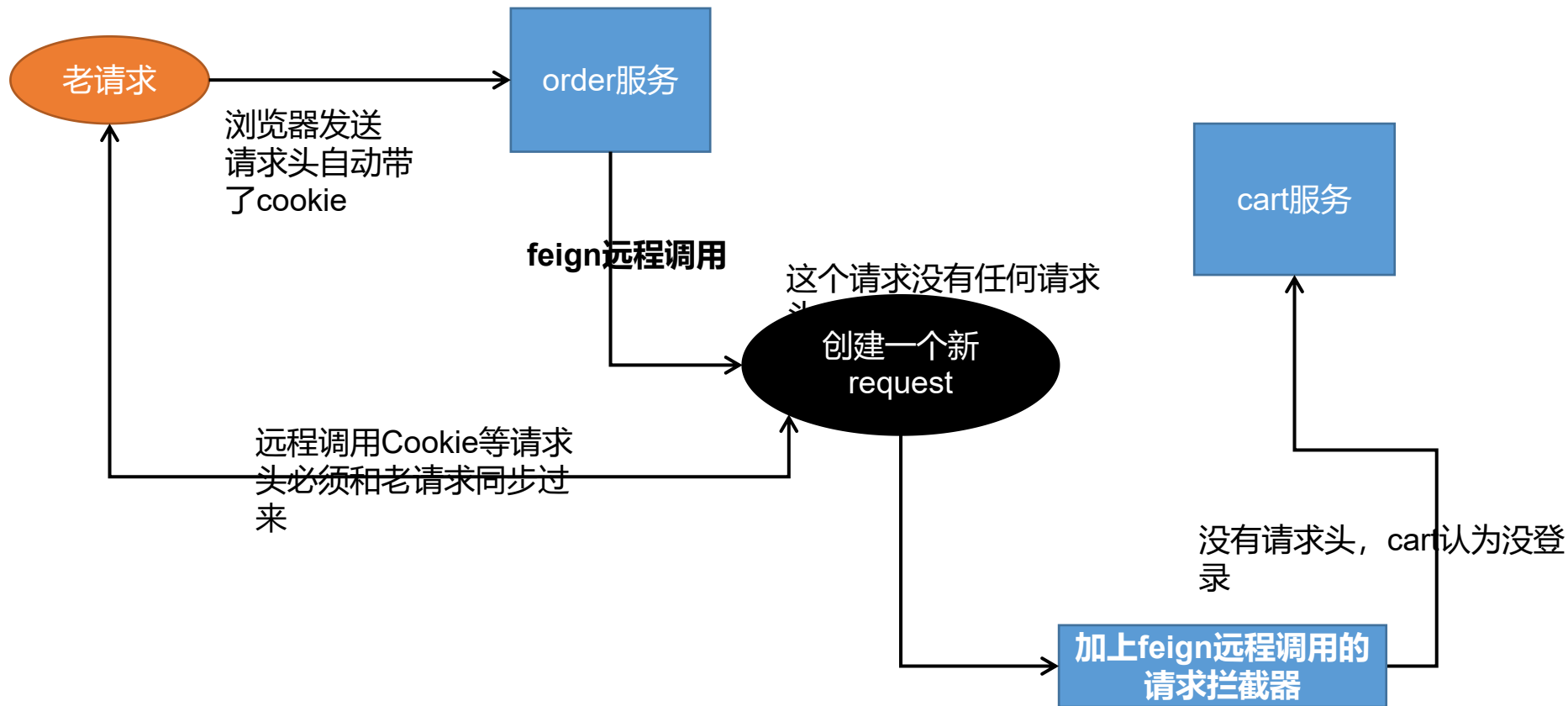


`Map<Thread, Object> threadLocal`



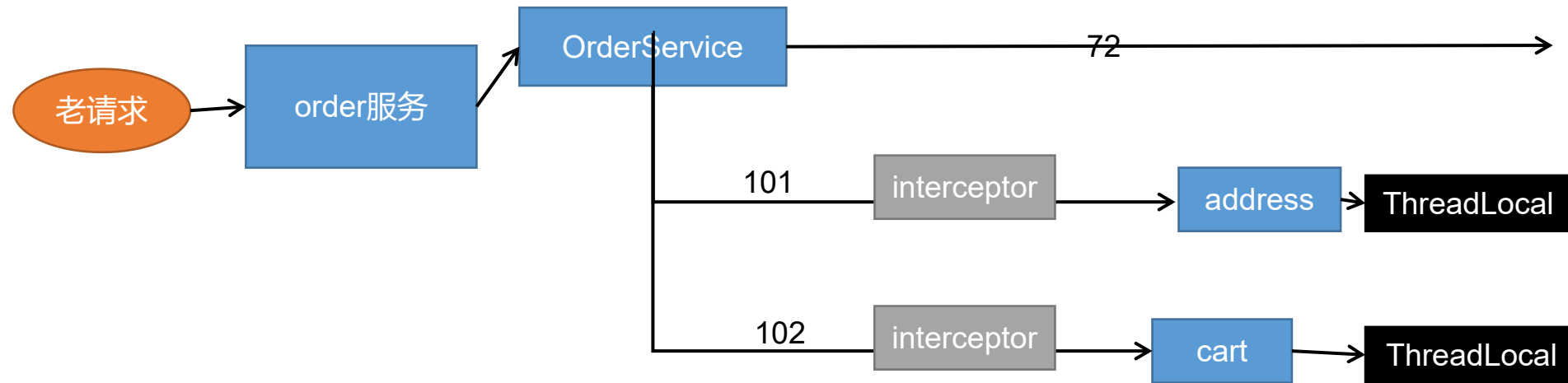


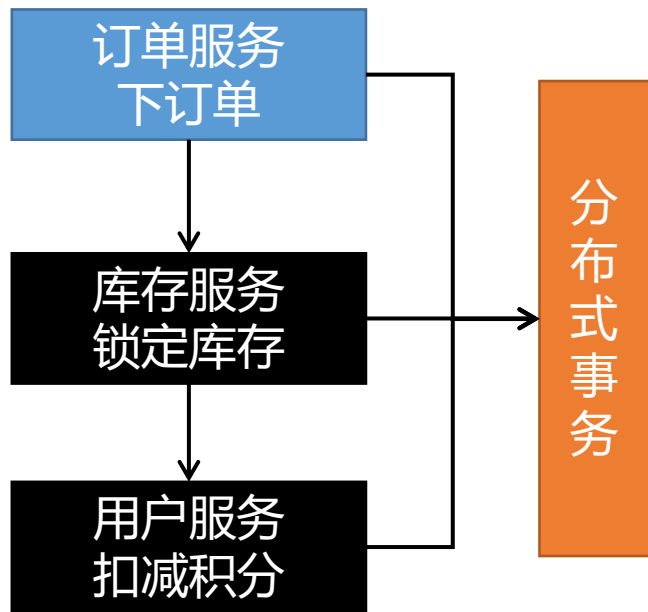
Feign远程调用丢失请求头问题





Feign异步情况丢失上下文问题





事务保证：

- 1、订单服务异常，库存锁定不运行，全部回滚，撤销操作
- 2、库存服务事务自治，锁定失败全部回滚，订单感受到，继续回滚
- 3、库存服务锁定成功了，但是网络原因返回数据途中问题？
- 4、库存服务锁定成功了，库存服务下面的逻辑发生故障，订单回滚了，怎么处理？

利用消息队列实现最终一致

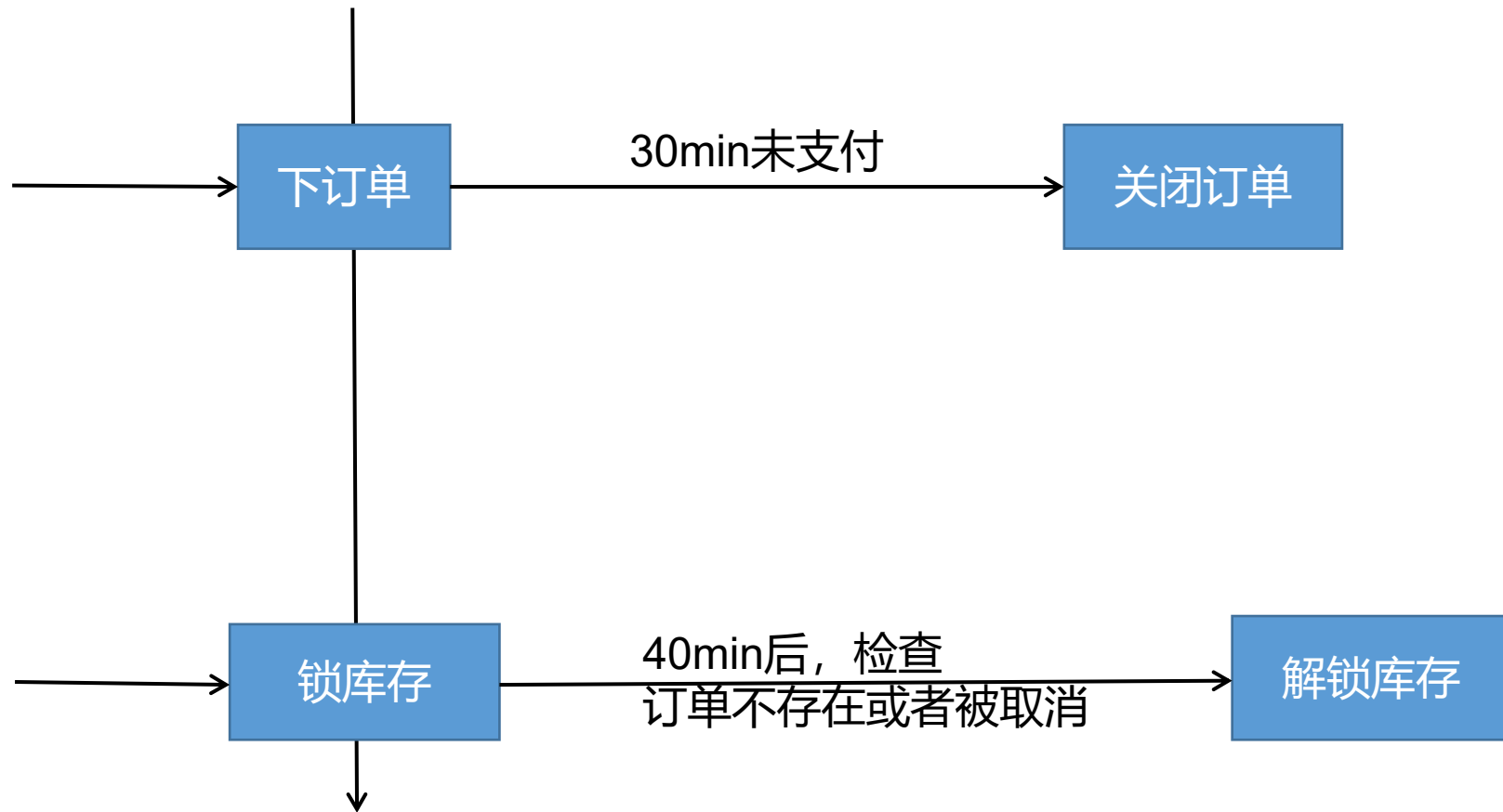
库存服务锁定成功后发给消息队列消息（当前库存工作单），过段时间自动解锁，解锁时先查询订单的支付状态。解锁成功修改库存工作单详情项状态为已解锁

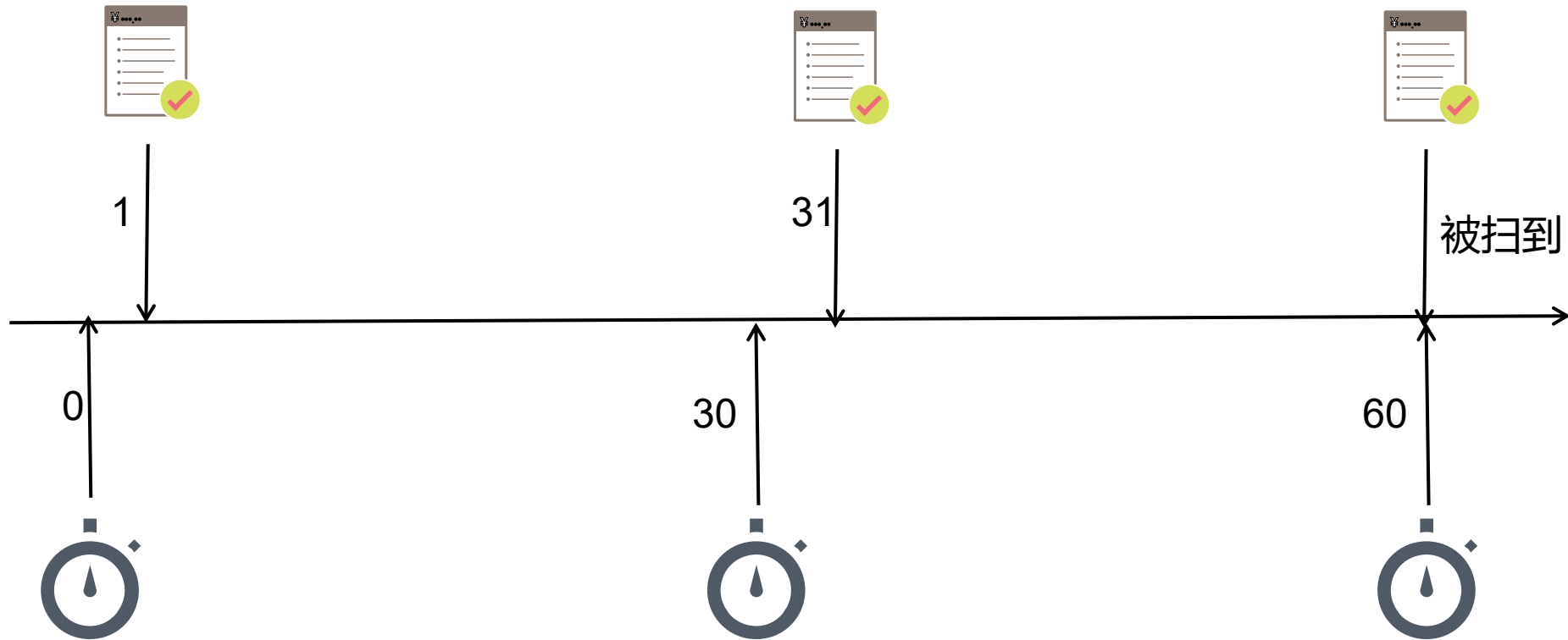
1、远程服务假失败：

远程服务其实成功了，由于网络故障等没有返回
导致：订单回滚，库存却扣减

2、远程服务执行完成，下面的其他方法出现问题

导致：已执行的远程请求，肯定不能回滚







订单释放&库存解锁

订单创建成功

订单解锁

库存锁定

库存解锁

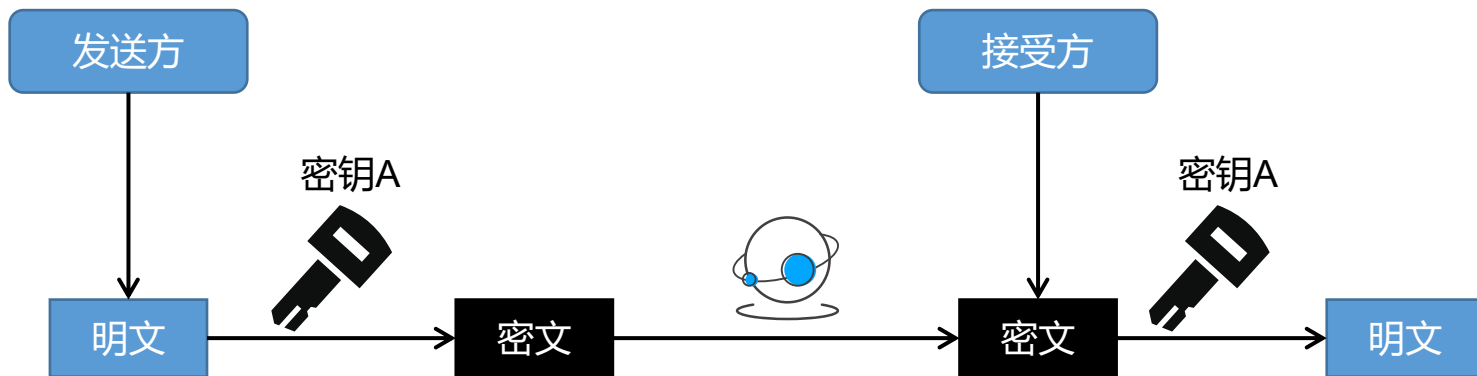
订单创建成功

订单解锁

库存锁定

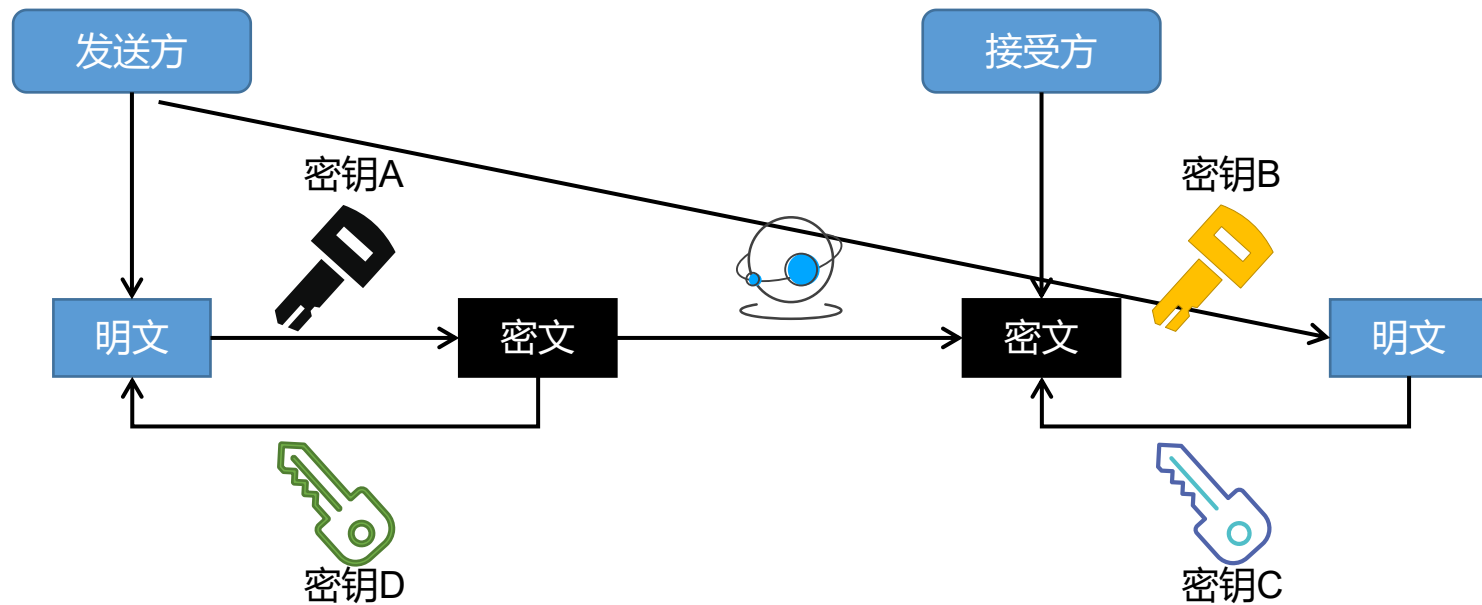
库存解锁

机器卡顿，消息延迟...



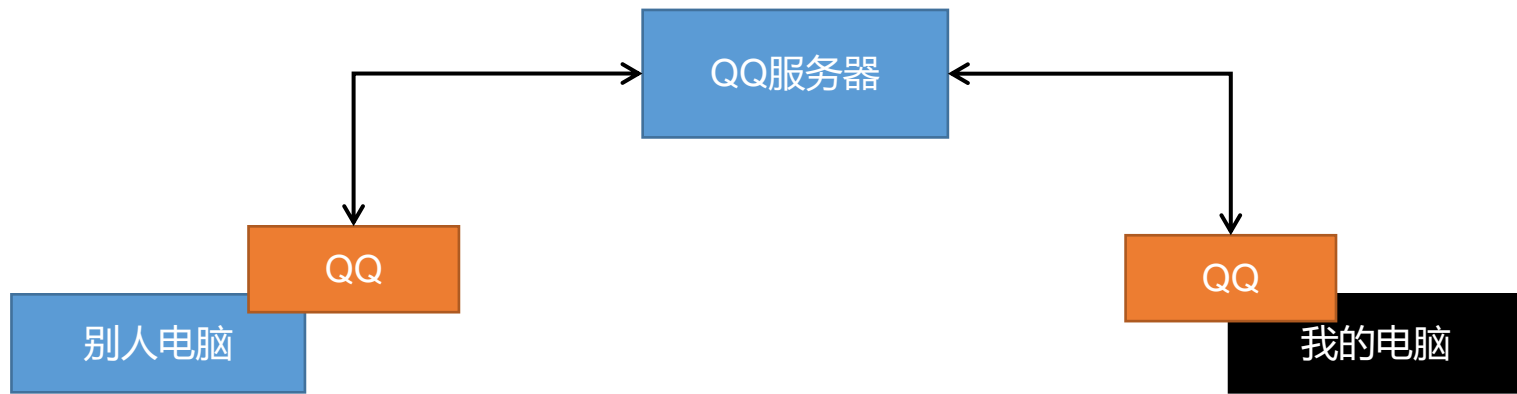
DES、3DES (TripleDES) 、AES、RC2、RC4、RC5和Blowfish等

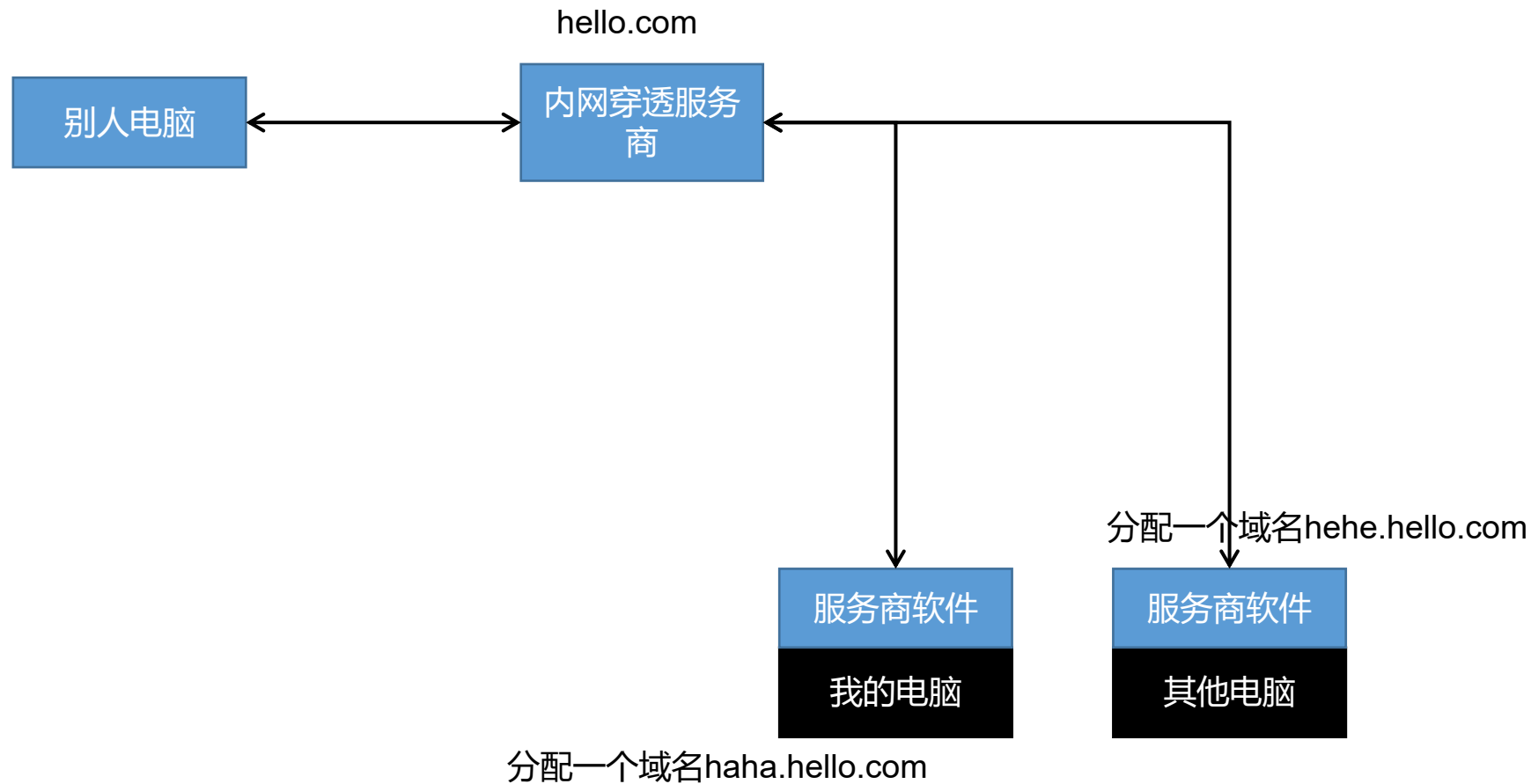
加密解密使用同一把钥匙

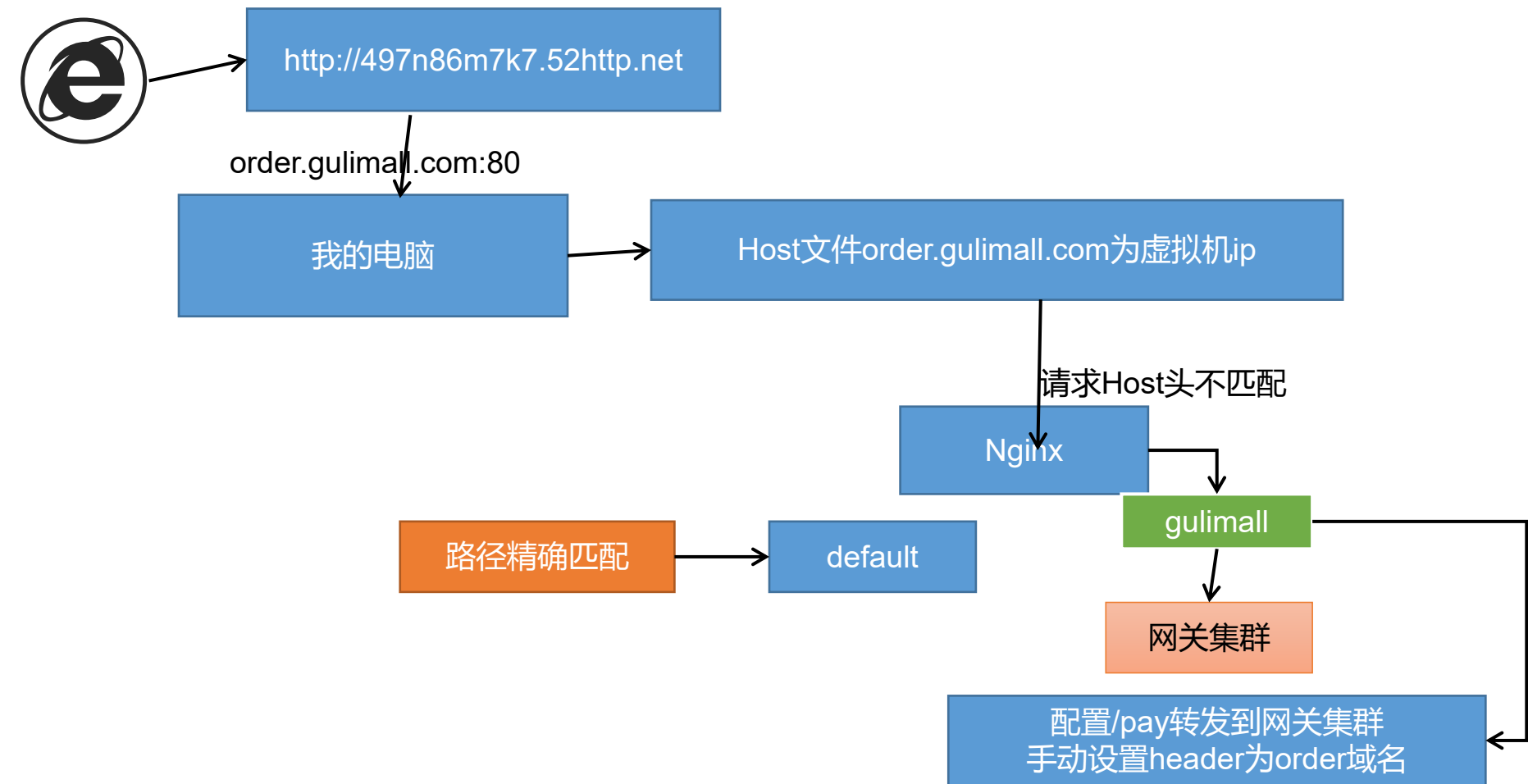


RSA、Elgamal等

加密解密使用不同钥匙





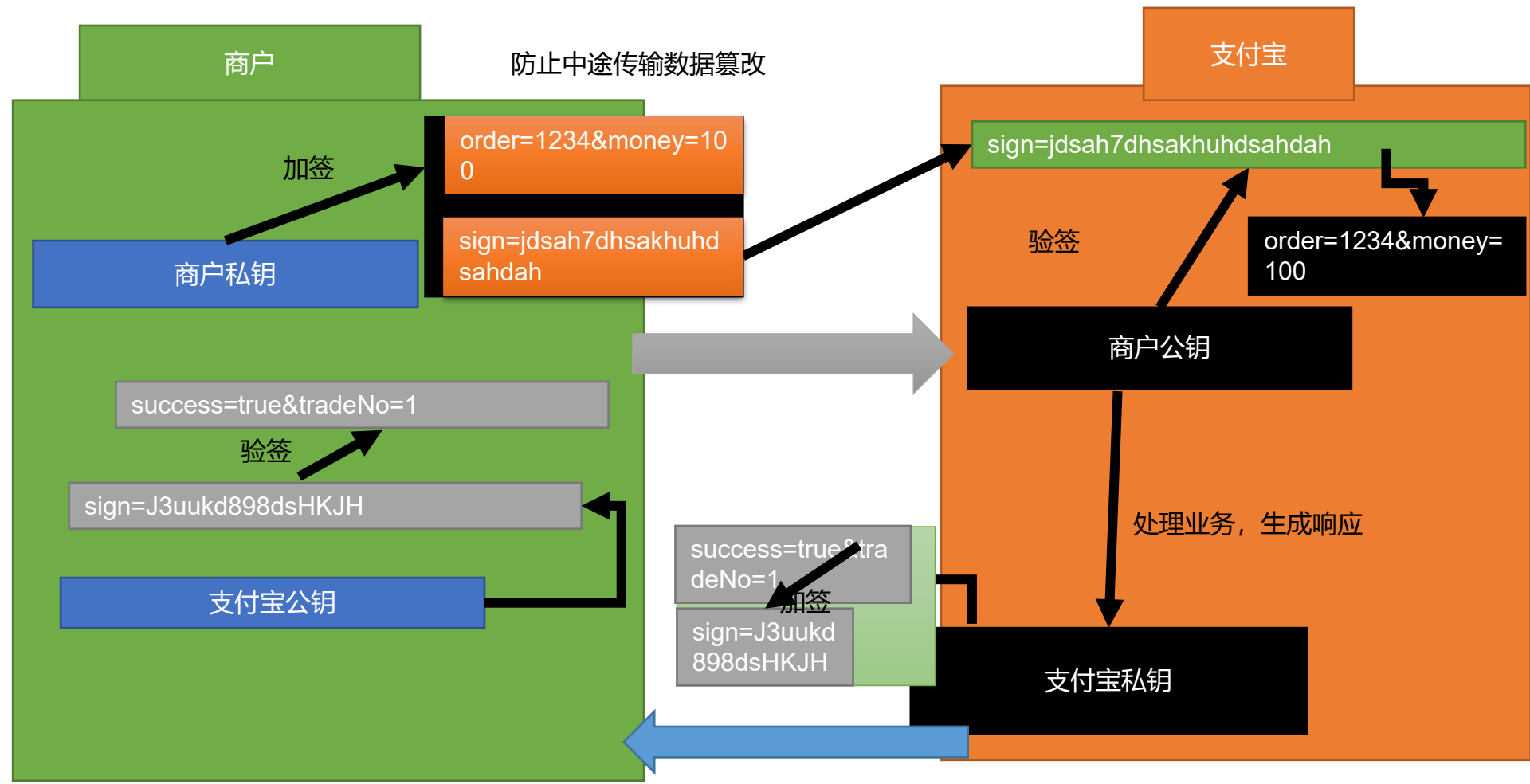




```
listen    80;

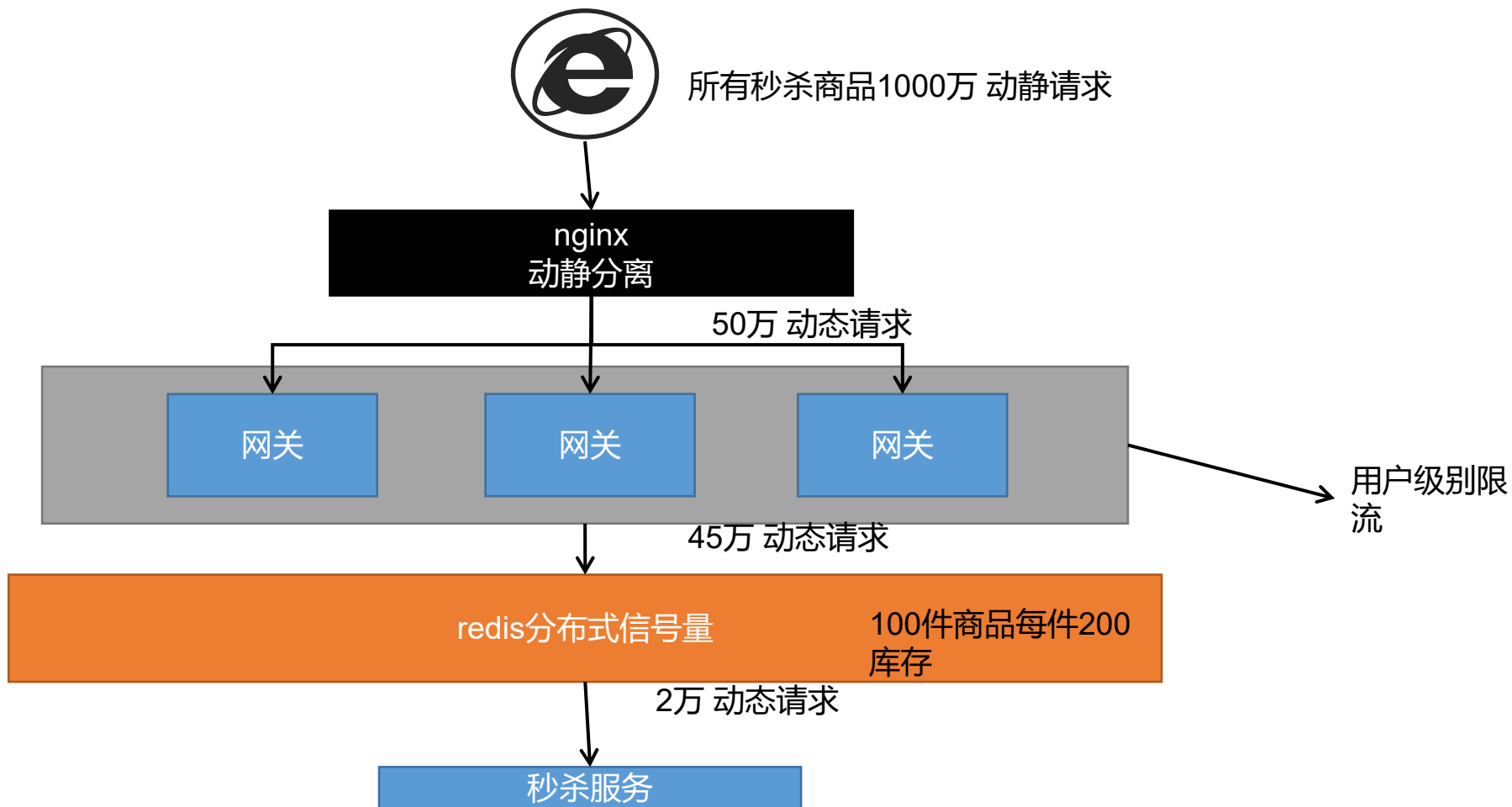
server_name  gulimall.com *.gulimall.com 497n86m7k7.52http.net;

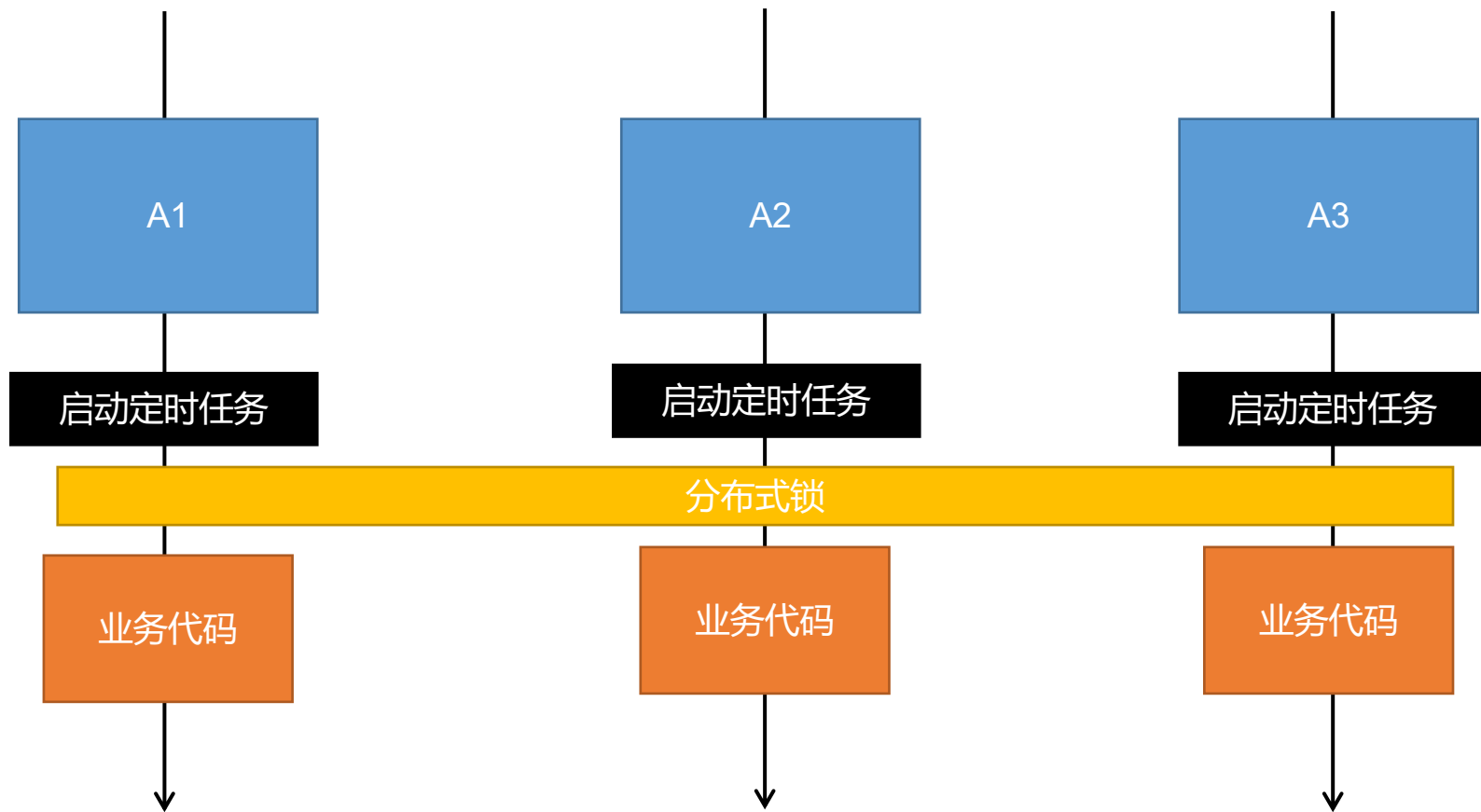

#charset koi8-r;
#access_log  /var/log/nginx/log/host.access.log  main;
location /static/ {
    root  /usr/share/nginx/html;
}
location /payed/ {
    proxy_set_header Host order.gulimall.com;
    proxy_pass http://gulimall;
}
```





- 1、订单在支付页，不支付，一直刷新，订单过期了才支付，订单状态改为已支付了，但是库存解锁了。
 - 使用支付宝自动收单功能解决。只要一段时间不支付，就不能支付了。
- 2、由于时延等问题。订单解锁完成，正在解锁库存的时候，异步通知才到
 - 订单解锁，手动调用收单
- 3、网络阻塞问题，订单支付成功的异步通知一直不到达
 - 查询订单列表时，ajax获取当前未支付的订单状态，查询订单状态时，再获取一下支付宝此订单的状态
- 4、其他各种问题
 - 每天晚上闲时下载支付宝对账单，一一进行对账







01

服务单一职责+ 独立部署

秒杀服务即使自己扛不住压力，挂掉。
不要影响别人

02

秒杀链接加密

防止恶意攻击，模拟秒杀请求，1000次/s攻击。
防止链接暴露，自己工作人员，提前秒杀商品。

03

库存预热+ 快速扣减

秒杀读多写少。无需每次实时校验库存。我们库存预热，放到redis中。信号量控制进来秒杀的请求

04

动静分离

nginx做好动静分离。保证秒杀和商品详情页的动态请求才打到后端的服务集群。
使用CDN网络，分担本集群压力



05

恶意请求拦截

识别非法攻击请求并进行拦截，网关层

06

流量错峰

使用各种手段，将流量分担到更大宽度的时间点。比如验证码，加入购物车

07

限流 & 熔断 & 降级

前端限流+后端限流

限制次数，限制总量，快速失败降级运行，熔断隔离防止雪崩

08

队列削峰

1万个商品，每个1000件秒杀。双11所有秒杀成功的请求，进入队列，慢慢创建订单，扣减库存即可。



A

缓存

B

异步

C

队排
好



谢谢观看