

微服务那些事儿

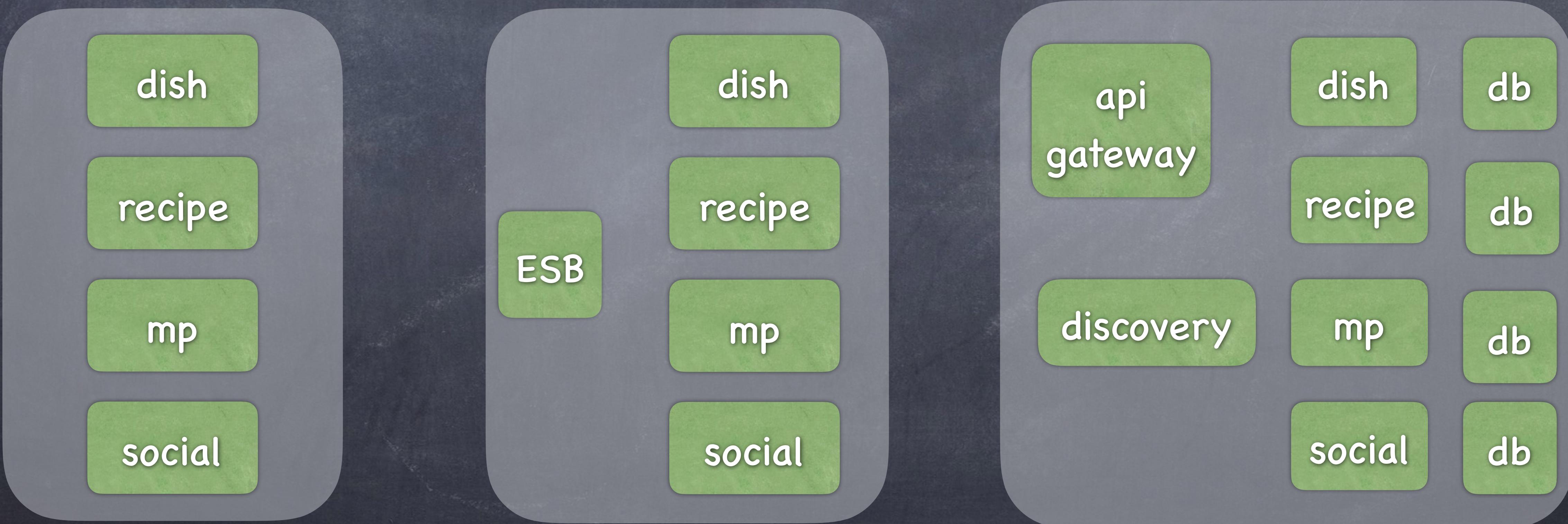
- xiaorui.cc

List

- ① 微服务
- ② 架构层
- ③ 缓存层
- ④ 数据层
- ⑤ 部署
- ⑥ 所谓经验

前世今生

一体化架构 >>> SOA服务化 >>> 微服务



micro service

- ① 微服务的优点
 - ② 松耦合，代码结构更加清晰
 - ② 开发者友好，避免老代码包袱.
 - ② 独立发布、快速迭代
 - ② 故障隔离
 - ② 增加重用，可组合
 - ② 针对性横向扩展

web design (宏)



dns

Nginx

Nginx

Nginx

micro service

micro service

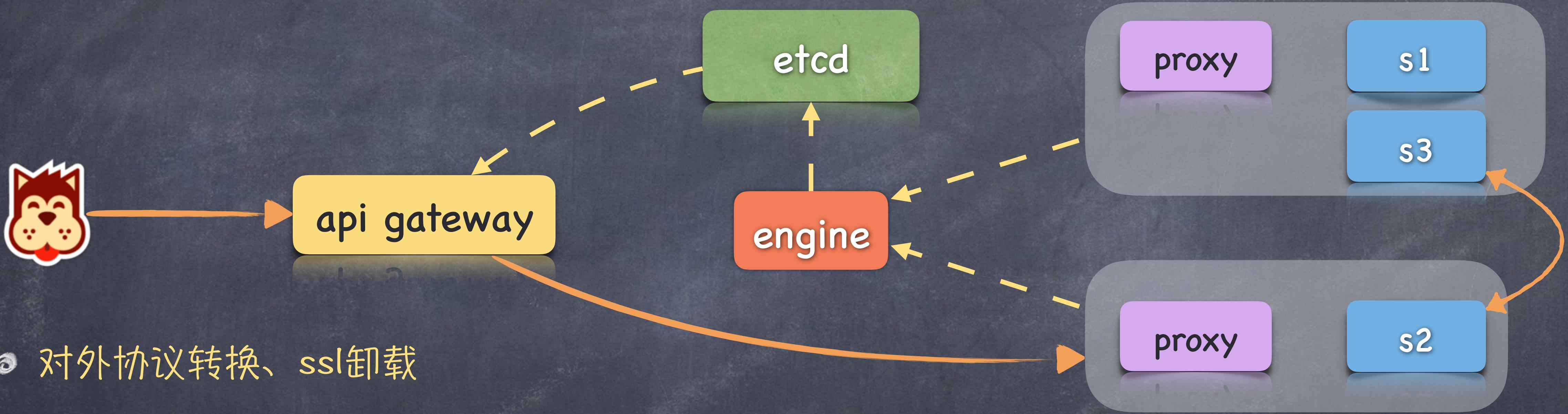
redis

redis

mysql

mysql

web design (微)



- 对外协议转换、ssl卸载
- 用户鉴权
- 服务发现及调度
- 熔断、降级、限频
- 收集服务注册信息
- 统计qps、error、success
- 分散gateway流量
- 无需重度开发sdk
- ...

协议组合选型

- ② rpc over tcp; 强约束rpc
- ② google protocol buffer
- ② apache thrift
- ② idl, 协议紧凑, 性能高
- ② rpc over http; 不约束rpc
- ② http msgpack
- ② 协议冗余, 简单, 自定义开发

协议组合

- ◎ 最终:
 - ◎ inside;
 - ◎ rpc over http
 - ◎ msgpack
 - ◎ outside;
 - ◎ http json

协议

{

```
"output_type": "json",
"call_func": "add_dish",
"args": [],
"kwargs": {},
"uid": "s123",
"login": true,
"timeout": 5,
"async": false,
"ip": "",
"sn": "",
"origin": "",
"app_id": "",
"timestamp": "",
"sign": ""
```



{

```
"code": "",
"error": "",
"res": {}
```



}

服务发现调度的演变

- ◎ 静态配置
- ◎ 智能dns调度
- ◎ 中心调度
- ◎ 基于调用方构建sdk

ALL == 伪命题 ???

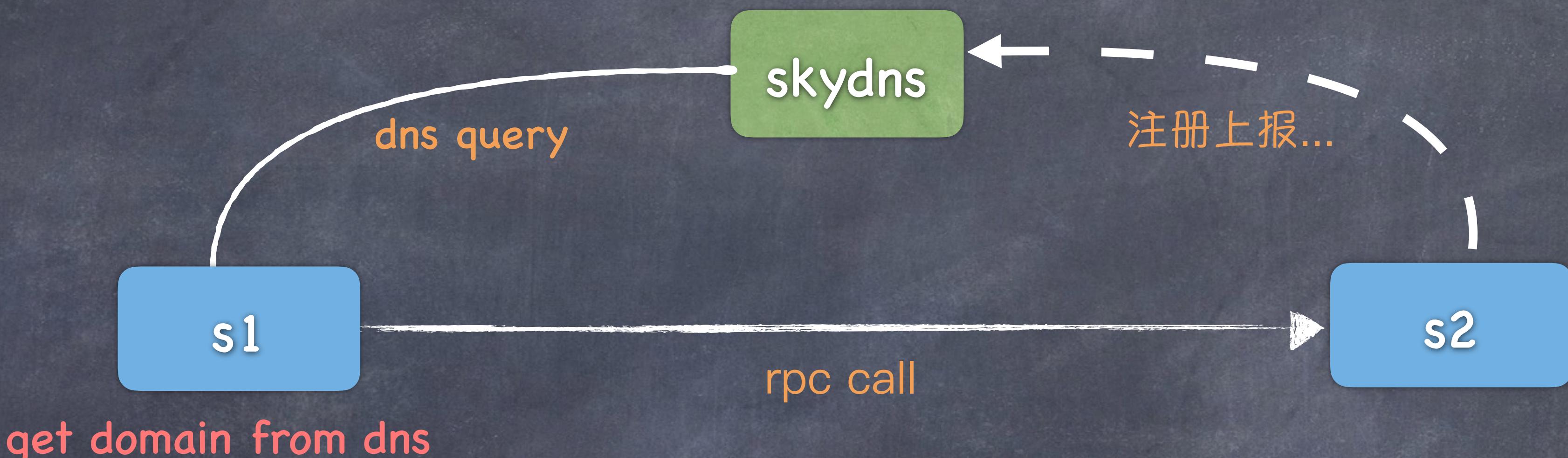
服务发现之静态配置



get hosts from config.ini;
random hosts

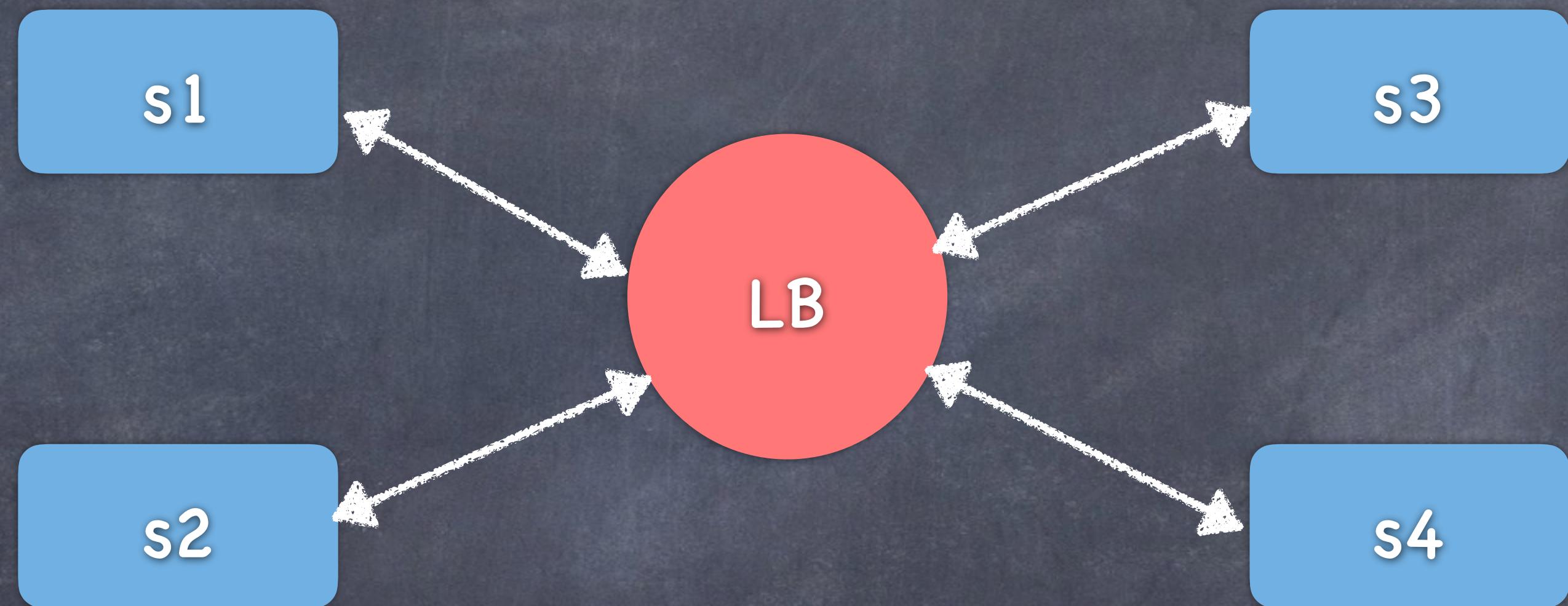
- ⌚ 频繁上线
- ⌚ 管理麻烦

服务发现之smart dns



- ⌚ 逃不出的ttl ?
- ⌚ 绕脑筋的A记录

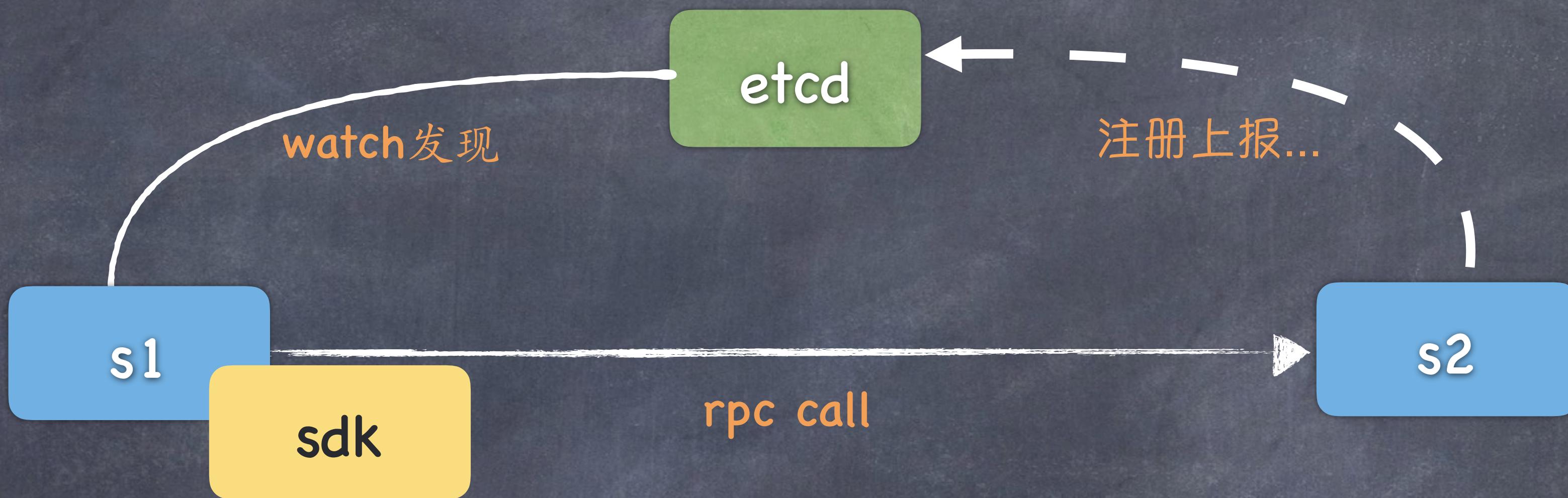
服务发现之负载均衡



- ④ 单点瓶颈 ?
- ④ 单点ha ?

- ④ LB hosts from etcd

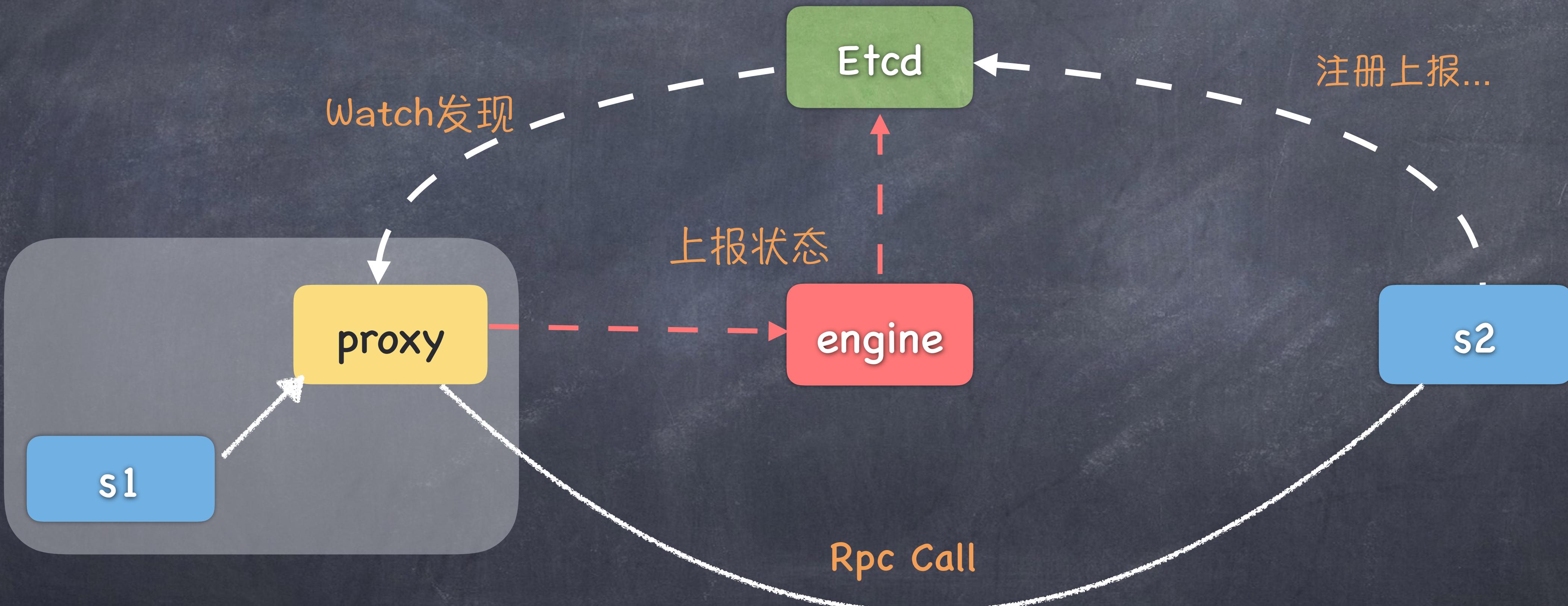
服务发现之sdk



④ 开发难度不简单？

④ 多种语言多次开发？

那么我们的选择？



future !!!

- ◎ 限频
- ◎ 熔断
- ◎ 降级



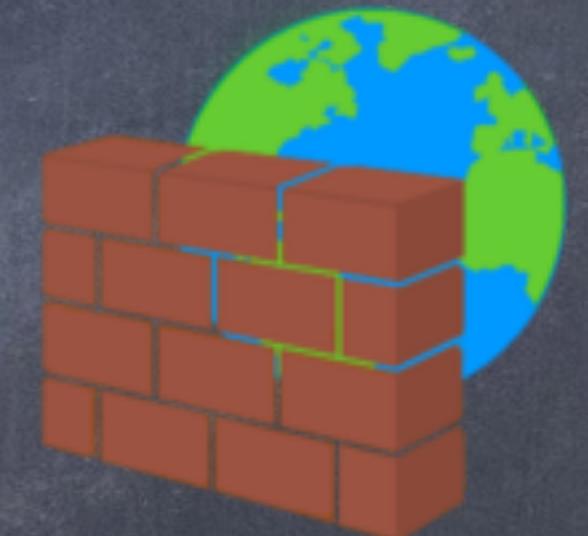
微服务下的session

- ① 共享存储 ?
- ② 逻辑复用并且调度器uid hash方案 ?
- ③ 每次访问user服务 ?
- ④ router check user login !



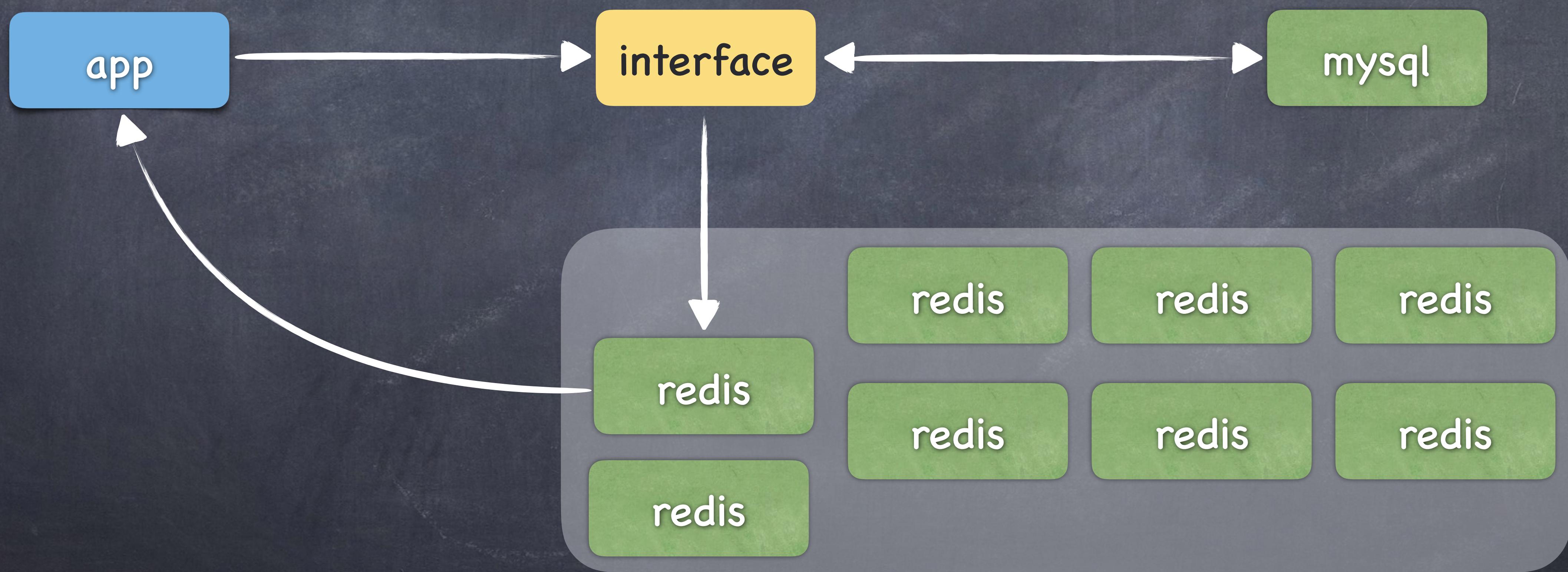
微服务下的接口安全

- ④ rsa + des
- ④ 慢，消息体加密
- ④ oauth2
- ④ 防串改及场景
- ④ sign (jwt)



```
HMACSHA256(  
    base64UrlEncode(body) + ".  
    timestamp + ".  
    SECCREATE_KEY + ".  
    XXX XXX  
)
```

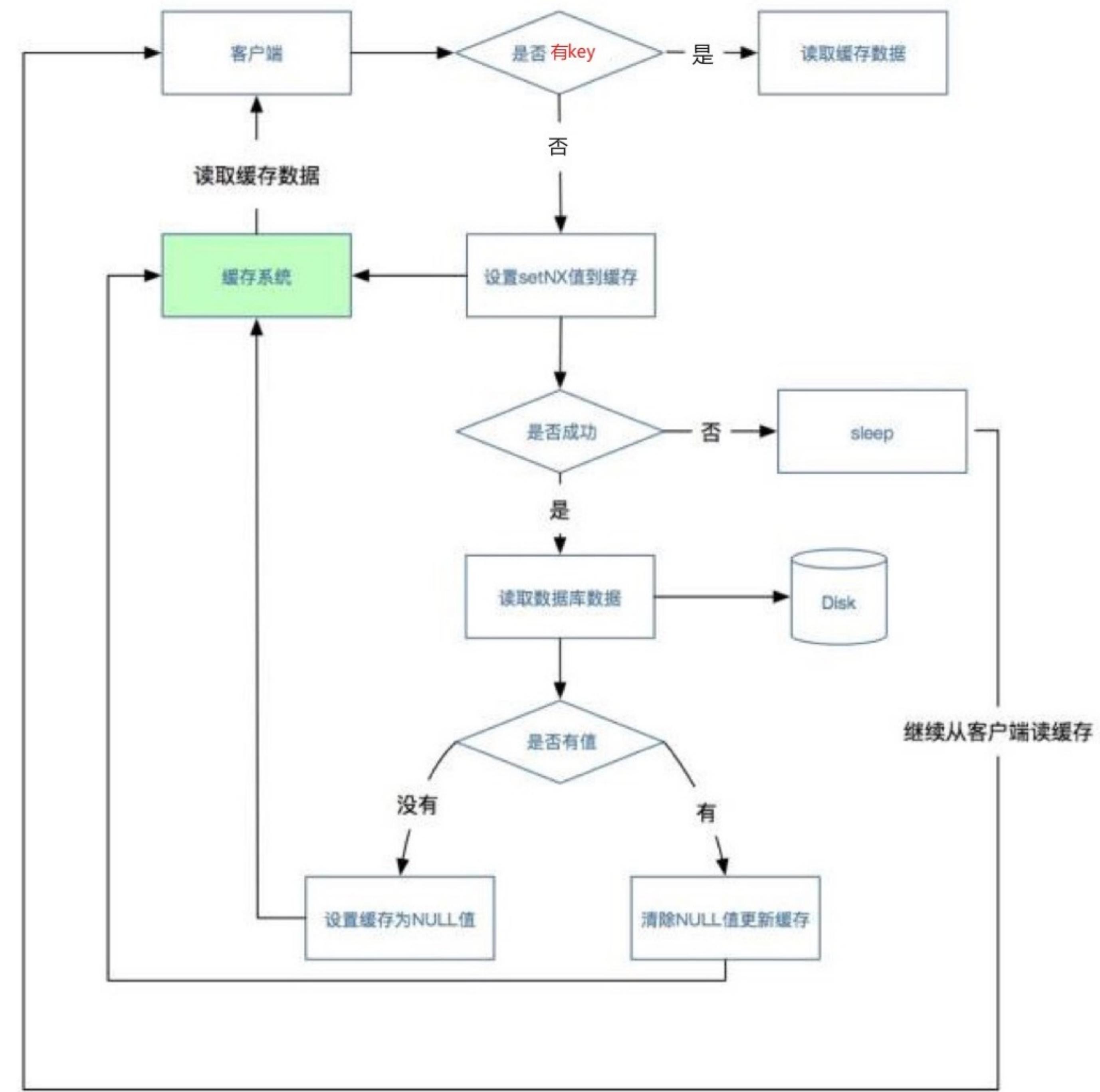
缓存层



缓存层

- 根据hot及 order by 定时服务主动推送热缓存
 - 当数据为空时，只有一人去db，其他人在服务端超时自旋
 - 热数据不采用expire，避免缓存穿透
 - 冷数据采用expire，避免缓存太多
- 多虚拟节点的一致性hash，升级缓存后减少cache miss

- ⑥ 一级缓存 (被干掉)
 - ⑥ 服务内部mmap + heap实现的ttl dict
- ⑥ 二级缓存
 - ⑥ redis多实例切分
 - ⑥ 解决单节点做rdb引起swap问题
 - ⑥ cpu单核问题
 - ⑥ 分布式扩展
- ⑥ 三级缓存
 - ⑥ 基于Rocksdb实现类redis数据结构



数据层

- ◎ 服务切了，数据当然也要独立切分
- ◎ 分库分表大势所趋
- ◎ 开启多实例，按照业务类型选择不同的**disk**实例
- ◎ 不使用中间件，自己开发**sql**路由

常见的分库分表

- ◎ 数据过度集中，不适用balance
 - ◎ 时间range
 - ◎ id range
- ◎ mode引起扩展不易
 - ◎ key hash
- ◎

分库分表问题

- ⑥ 分库分表如何做到高效的索引查询？
- ⑥ 数据冗余表如何保证数据完整？

场景1

| 帖子id | author |
|-----------|--------|
| gtid+user | 张三 |

既想通过author查询，又想通过tiezi_id，如何分库分表？

按照author分库分表，在生成tiezi_id时加入分表标志位！

场景2

| tiezi_id | author | pub_date |
|-----------|--------|------------|
| gtid+user | 张三 | 2017-05-21 |

查询方式 或 **tid**, 或 **author**, 或 **pub_date**, 该如何分库分表?

按照**author**分库分表，在**tiezi_id**加入分表标志位 **or author** !

+

冗余表：按照**pub_date**分库分表，加入**tiezi_id** 及 **author**信息

场景3

| tiezi_id | author | pub_date | update_date |
|-----------|--------|------------|-------------|
| gtid+user | 张三 | 2017-05-21 | 2017-05-30 |

查询方式 (`tid, author, pub_date, update_date`), 该如何分库分表?

在场景2的基础上, 再加入`update_date`的分表?

如果`update_date`的需求小于10%? 那么多表轮询 !!!

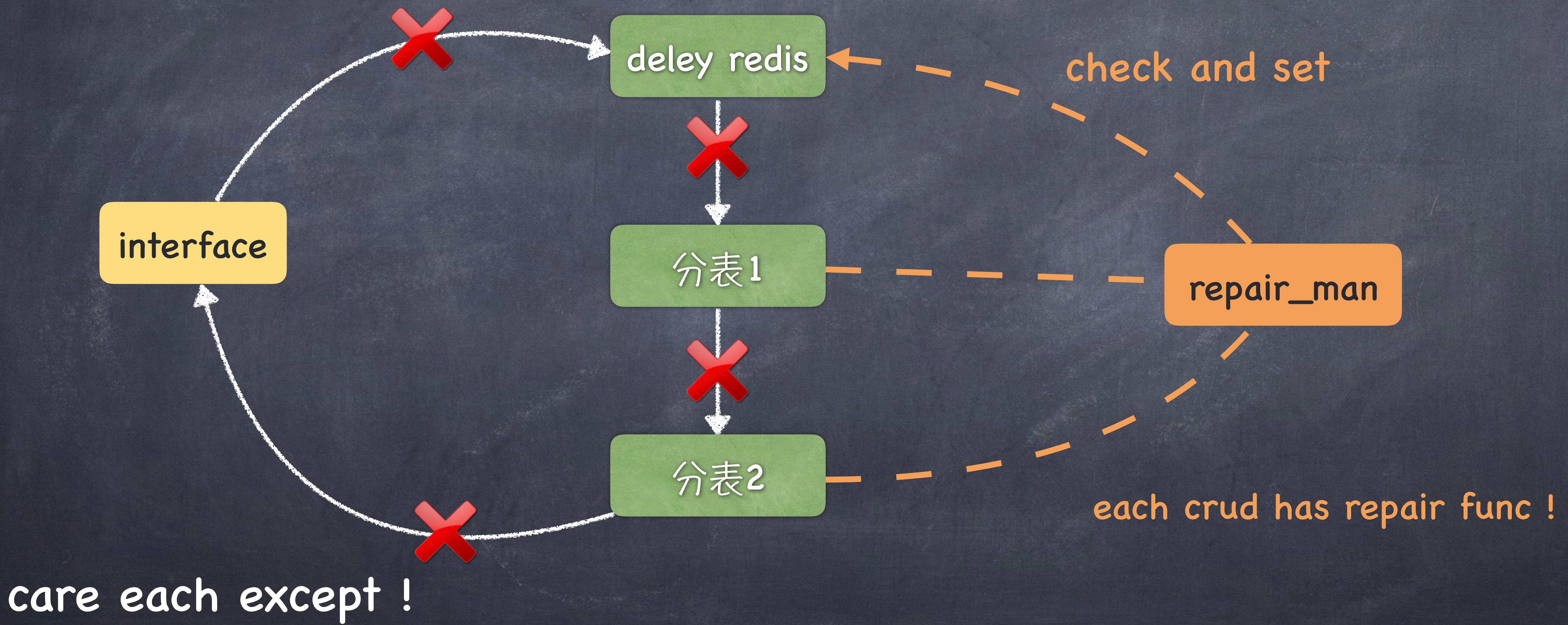
场景4

| user | target_user | ... |
|------|-------------|-----|
| 峰云 | 李宇春 | ... |

查询方式 或 user, 或 target_user, 该如何分库分表?

解决方法，两个维度的分库分表

repair



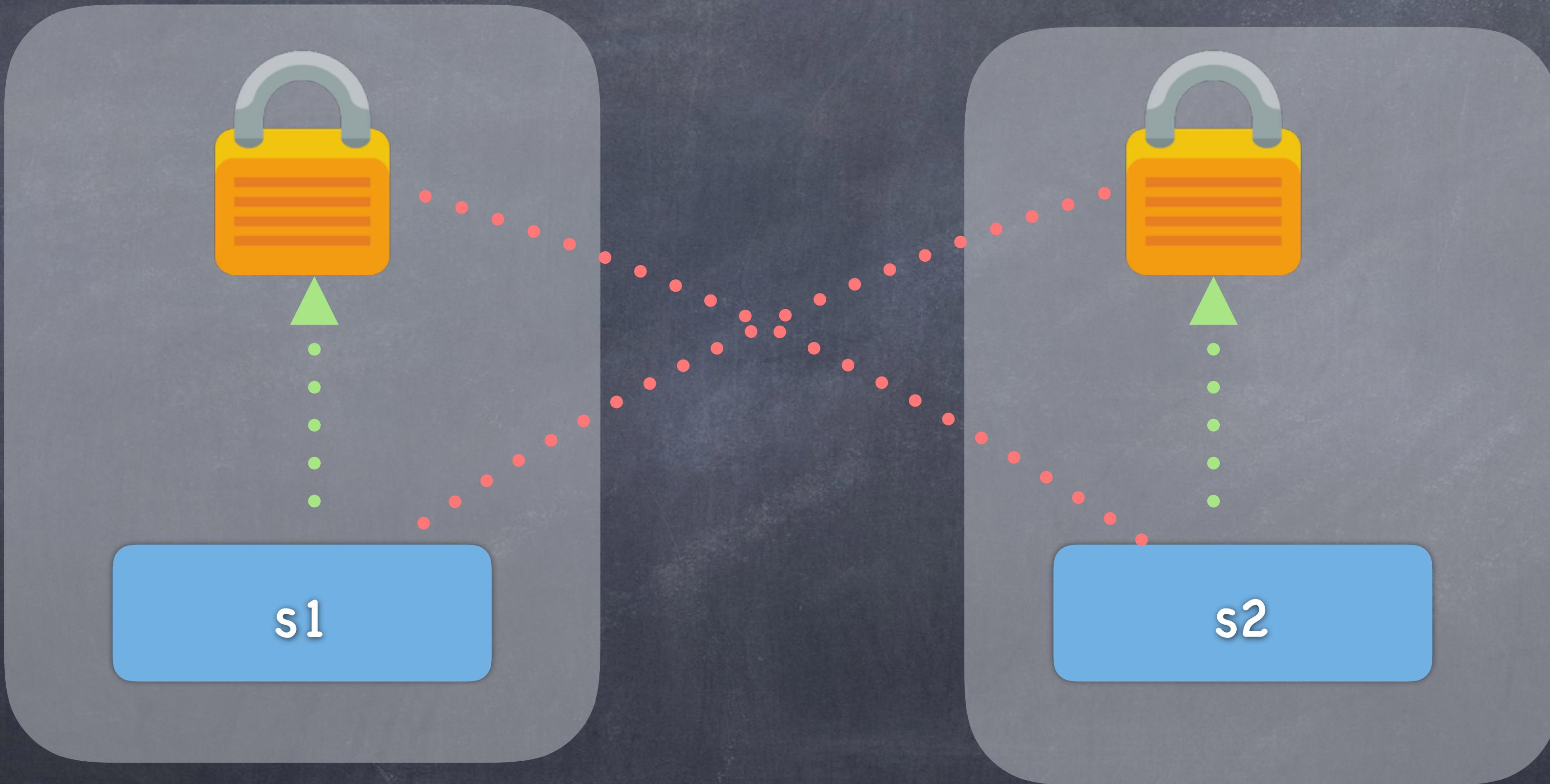
事务？

- ◎ 事务相关的表尽量在一个db库中
- ◎ 强一致性, 2pc, 3pc
- ◎ 太耗时...
- ◎ 消息队列最终一致性
- ◎ 高效, 易理解
- ◎ 读多写少场景用cas

消息队列处理事务



分布式事务中的死锁



问题

- ◎ 分布式id
- ◎ 伪自增；独立库分配auto_id
- ◎ 客户端生成；snowflake
- ◎ hash节点不够，扩节点？
- ◎ 预先设立16/ 32 /64 /128 hash值
- ◎ 外键？ 干掉!!! 分布式系统中杜绝外键 !!!

metrics 性能监控

- ◎ 分布式系统必备 !!!
- ◎ 框架自身的性能收集
- ◎ 函数级别探针收集
- ◎ db query消耗收集
- ◎





调试层 (tracing)

- ④ gateway 自动嵌入 trace_id
- ④ 每个逻辑块、服务区一个 span_id
- ④ span 之间存在父子调用链关系
- ④ 根据 trace_id 追踪问题
- ④ 根据 span 输出全链路的调用关系及时间轴

simple tracing



业务性能优化

- ◎ 批量io
- ◎ 长连接
- ◎ 缓存
- ◎ 异步io框架
- ◎

deploy

1. 打包前的全局单元测试
2. docker build image
3. 灰度发布docker image
4. sentry及报表追踪异常

testing

1. 单元测试
2. 推送到闭环成套的测试环境
3. 从线上抽样请求到测试环境

异地多活

- ◎ 重点在于 DB
- ◎ 解决idc数据同步问题
- ◎ 备idc部分可写及重要只读
- ◎ 通过消息队列同步数据
- ◎ dns层切换数据中心



异地多活

采用多种手段，保证绝大部分用户的核心业务异地多活！

- ◎ 双主
- ◎ 专线？
- ◎ 同城 $\leq 1\text{ms}$
- ◎ 跨城 $\geq 1\text{ms}$
- ◎ 逃不掉的分区脑裂
- ◎ 半同步？



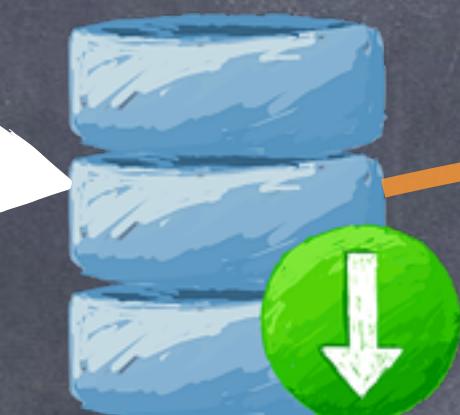
异地多活

北京兆维



- user login
- user create
- user update

m-s async data



北京亦庄

scan binlog insert



- user login
- user create
- user update

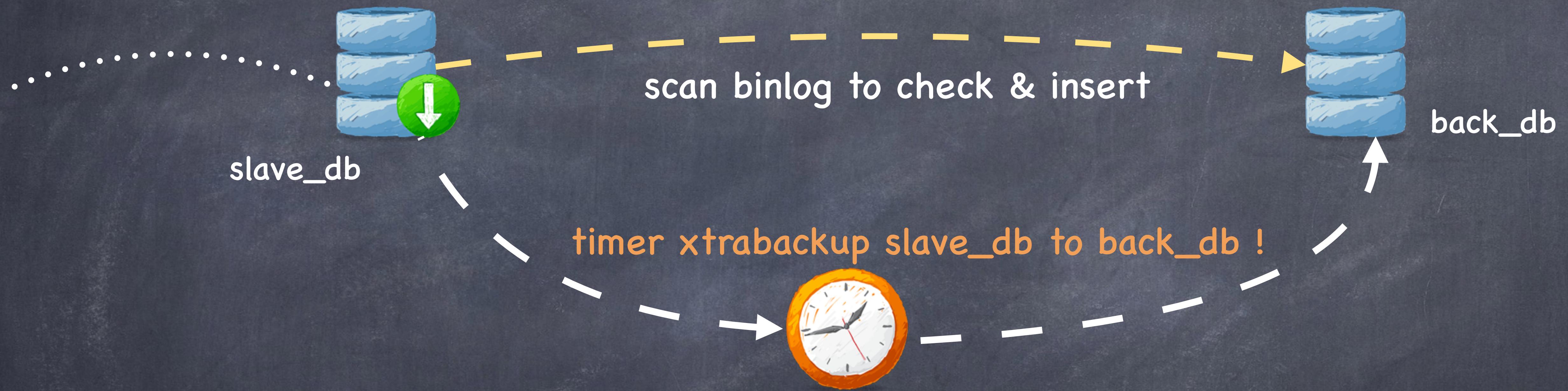


msg

ack



伪同步 (备)



- ④ 为什么会有**slave**及**back**的角色? 为什么不直接写**slave**!
- ④ 避免写坏主从关系
- ④ 重置数据的作用在于, 粗暴的重新平衡数据

异地多活

- ◎ 不要陷入完美主义
- ◎ 啥都想着同步
- ◎ 保证备数据中心的核心业务存活
- ◎ 选择性只同步非账号及非金钱的数据
- ◎ 提高用户体验
- ◎ 应用层控制mq消息的可靠性



“ Q&A ”

- 峰云就她了