

# 支撑百度搜索引擎 99.995%可靠性服务发现 的设计

百度 郑然

技术架构未来

## 自我介绍



- 百度网页搜索部 – 架构师
- 七年搜索引擎架构工作
  - 流式索引计算系统
  - 离线计算平台架构
  - 大规模在线服务治理
- 百度C++标准委员会委员
- 分布式系统 & 高可用架构

# 目录 CONTENTS

搜索引擎的挑战

服务发现的原理

关键技术与实践难点

典型应用案例

总结和思考



# 搜索引擎的挑战

## 挑战

机器数量多， 服务数量大

数万台服务器， 数十万个服务， 分布在多个IDC

服务变更多， 变更数据大

每天几十万次变更， 每周10P量级的文件更新， 千余人并行开发上百个模块

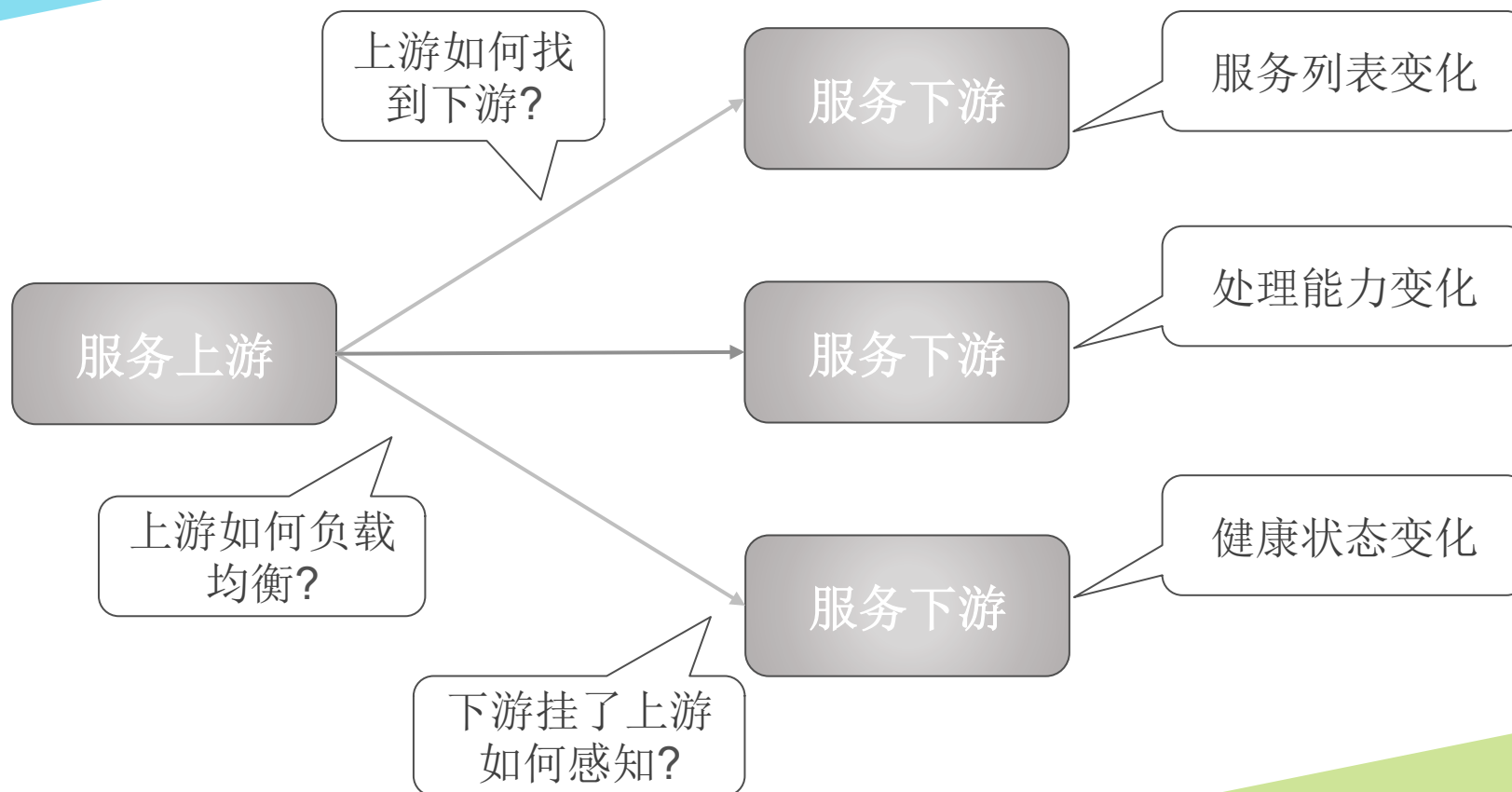
检索流量大， 稳定性要高

每秒数十万次请求， 满足99.995%的可用性， 极短时间的故障都可能引发大量的拒绝

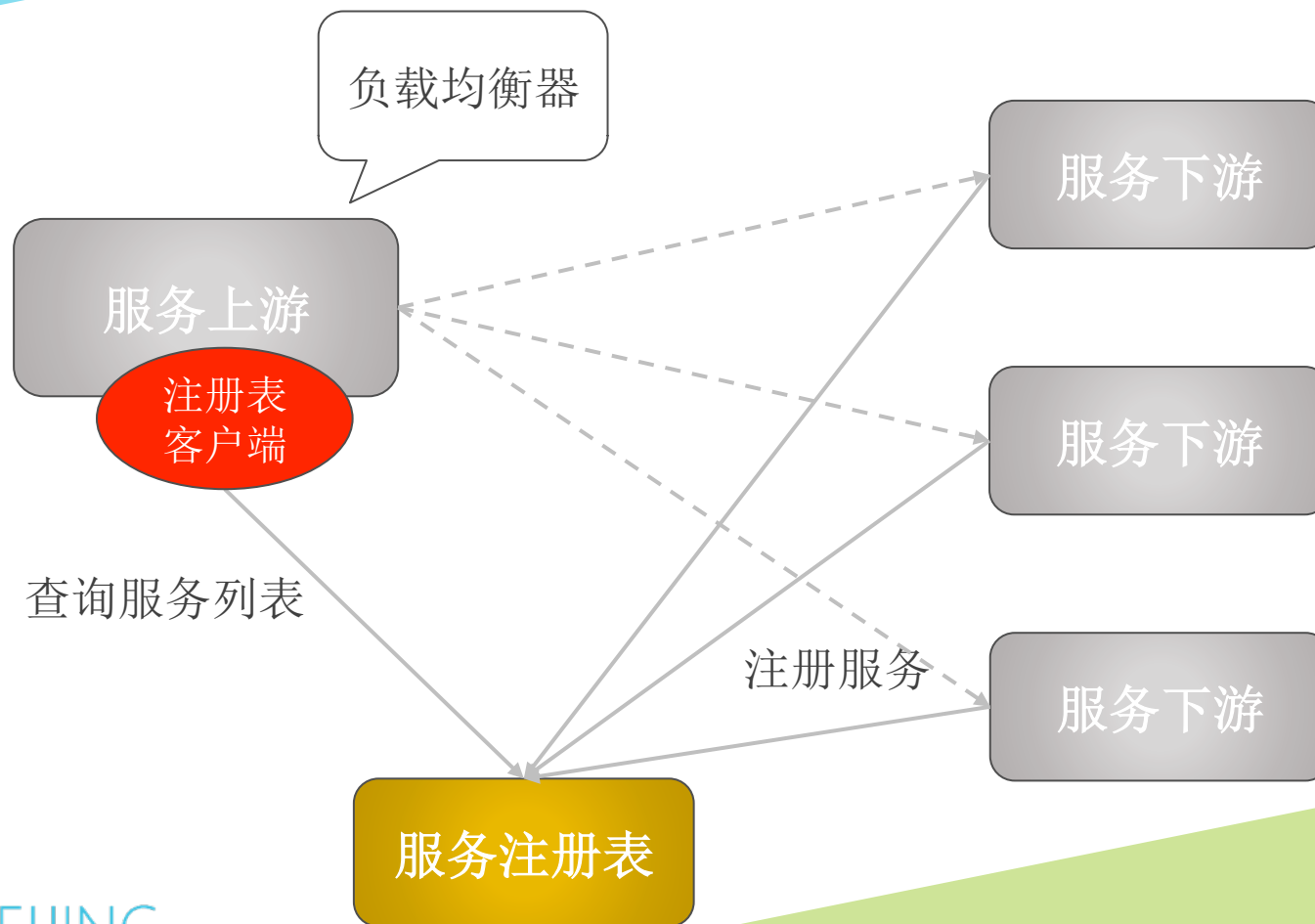


## 服务发现的原理

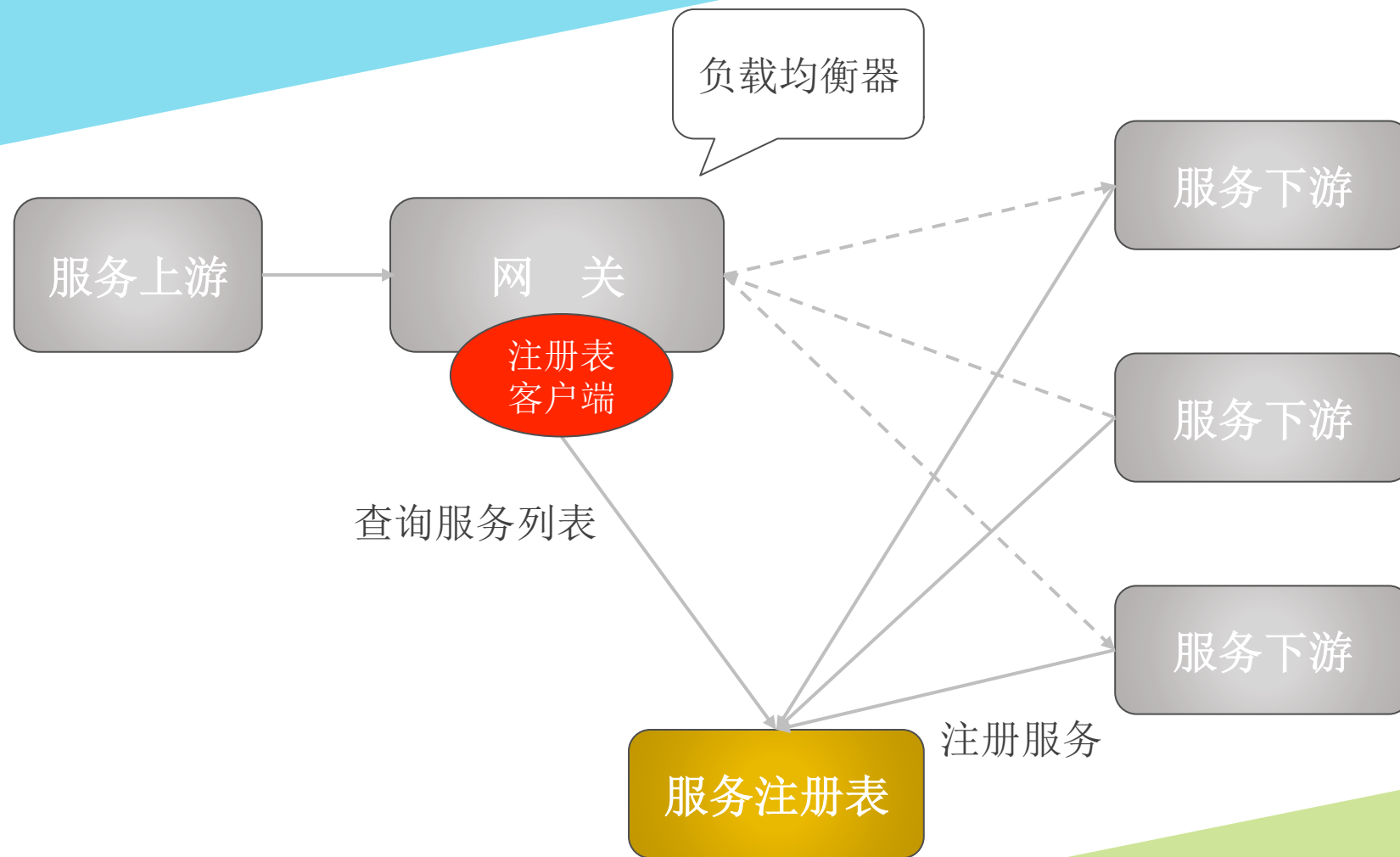
## 什么是服务发现











## 服务发现 组件

### 服务 注册表

- 分布式存储, 持久化服务地址和属性
- 服务名字全局唯一
- 支持增删改查
- 支持高并发高吞吐, 延迟要求不太高(几十ms)
- 读多写少

### 注册表 客户端

### 负载 均衡

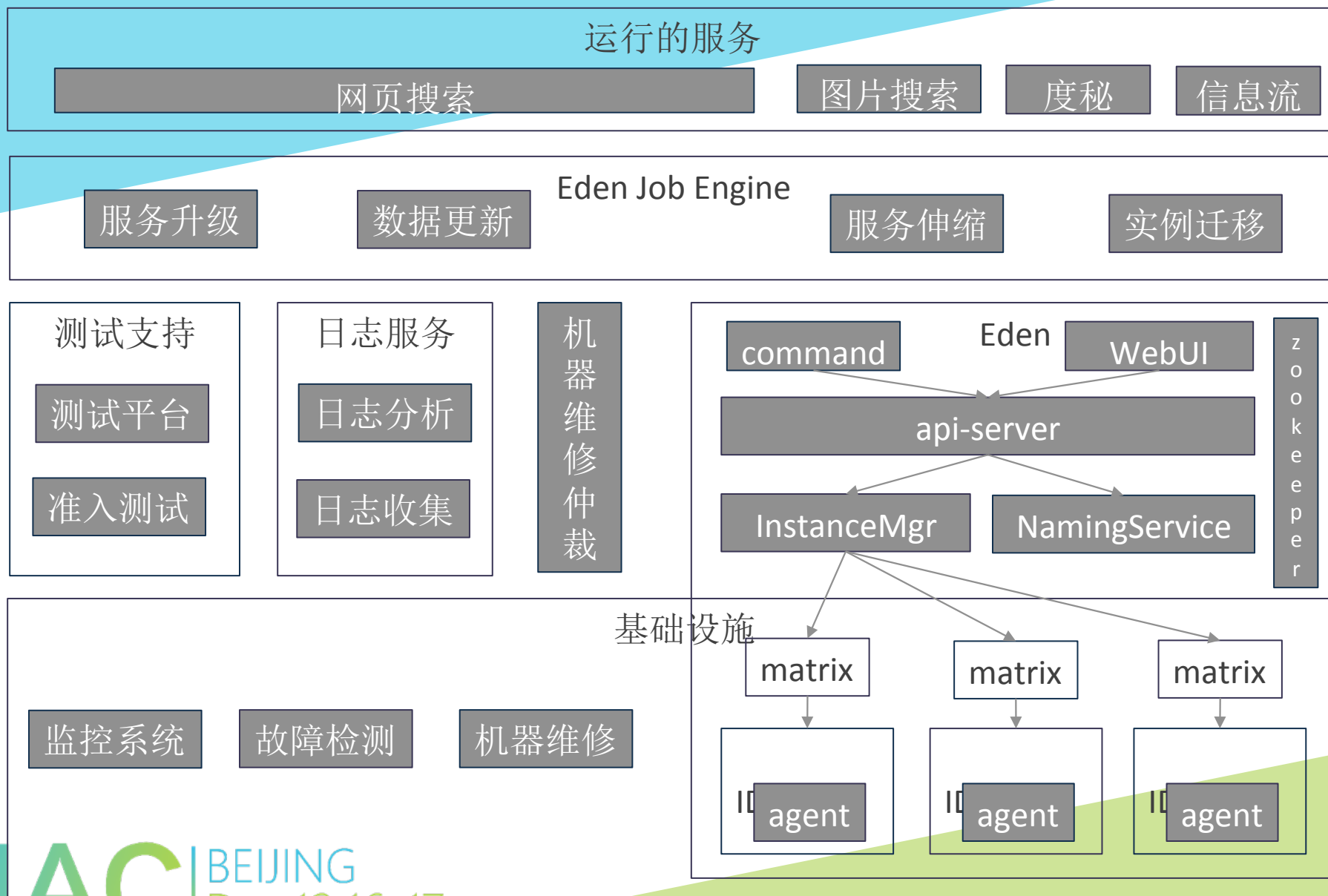
- 根据负载选择某个服务
- 故障服务的剔除和探活



## 关键技术与实践难点

# 可变服务      v.s.   不可变服务

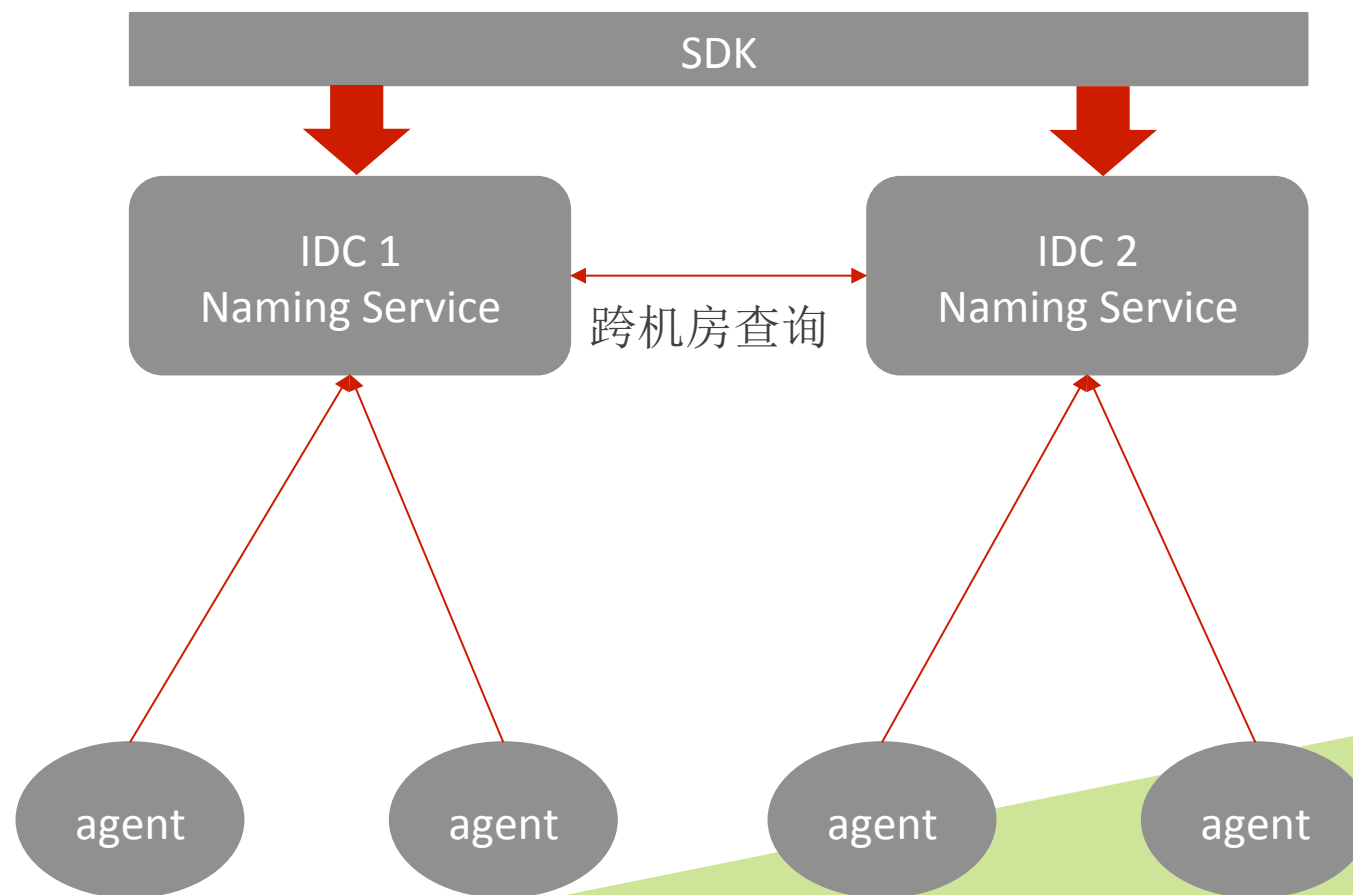
# Eden 架构蓝图

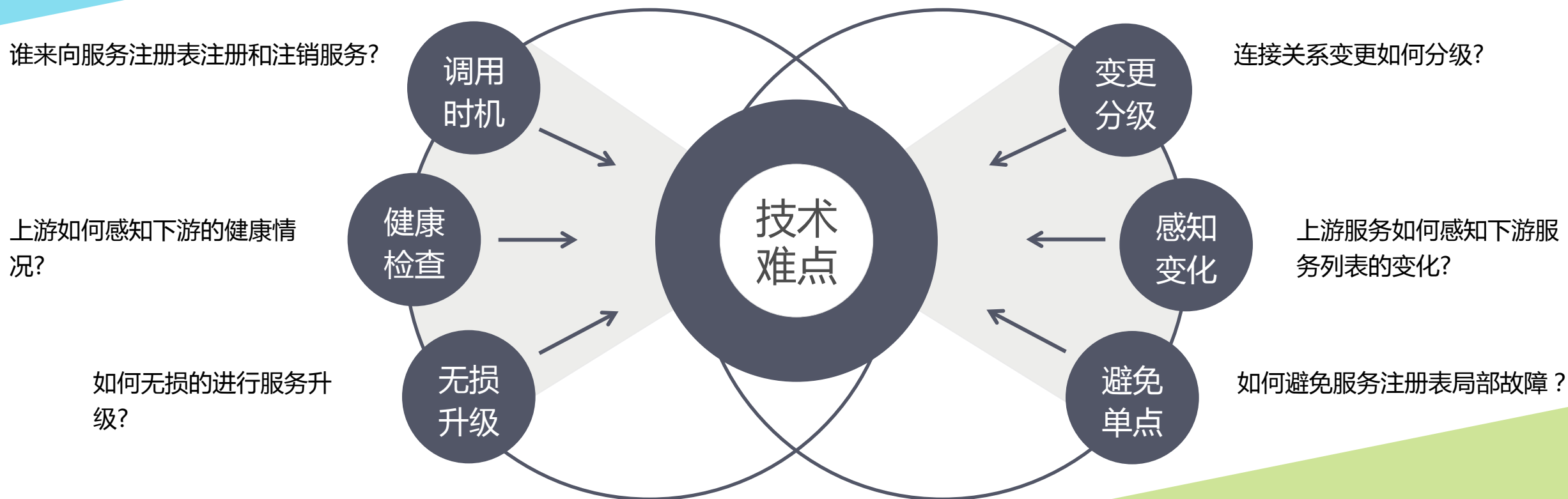


## 服务发现 架构

app\_id必须全局唯一  
app\_name.idc.zone

```
{,
"naming":{
  "default_ns_tag":{
    "ns_state":"NEW"
  },
  "default_visible":true,
  "dependency":[
    {
      "alias":"se_589",
      "app_id":"bs_se_589",
      "attribute":{
        "MIN_USABLE_RATIO":"90",
        "SVC_GROUP":"all_bs,se,se_589"
      },
      "version":"overlay_2016-11-23 17:19:371375344376"
    },
    {
      "alias":"se_590",
      "app_id":"bs_se_590",
      "attribute":{
        "MIN_USABLE_RATIO":"90",
        "SVC_GROUP":"all_bs,se,se_590"
      },
      "version":"overlay_2016-11-23 17:19:37241337247"
    },
    {
      "alias":"se_591",
      "app_id":"bs_se_591",
      "attribute":{
        "MIN_USABLE_RATIO":"90",
        "SVC_GROUP":"all_bs,se,se_591"
      },
      "version":"overlay_2016-11-23 17:19:3773166471"
    }
  ]
},
}
```







调用时机

服务注册表

服务

### 服务自己

服务启停强依赖注册表  
服务需要植入SDK  
干扰运维可预期性  
影响过载保护策略

agent

服务注册表

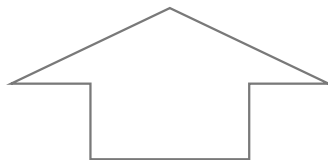
服务

### 第三方组件

添加删除时修改注册表  
不需要植入SDK  
弱依赖注册表  
更容易运维效果监控  
降低注册表负载

健康检查

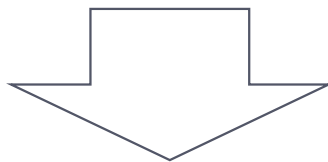
弱依赖注册表, 感知周期短, 准确性高



客户端和服务端建立心跳和探活机制



服务自己做健康检查, 汇报给注册表



强依赖注册表, 未必准确, 感知周期长

服务端健康检查

客户端健康检查

## 升级影响

升级就意味着同时重启的服务数量比常态多

## 失败概率大

上游服务访问下游服务失败的概率增加

## 限制性重试

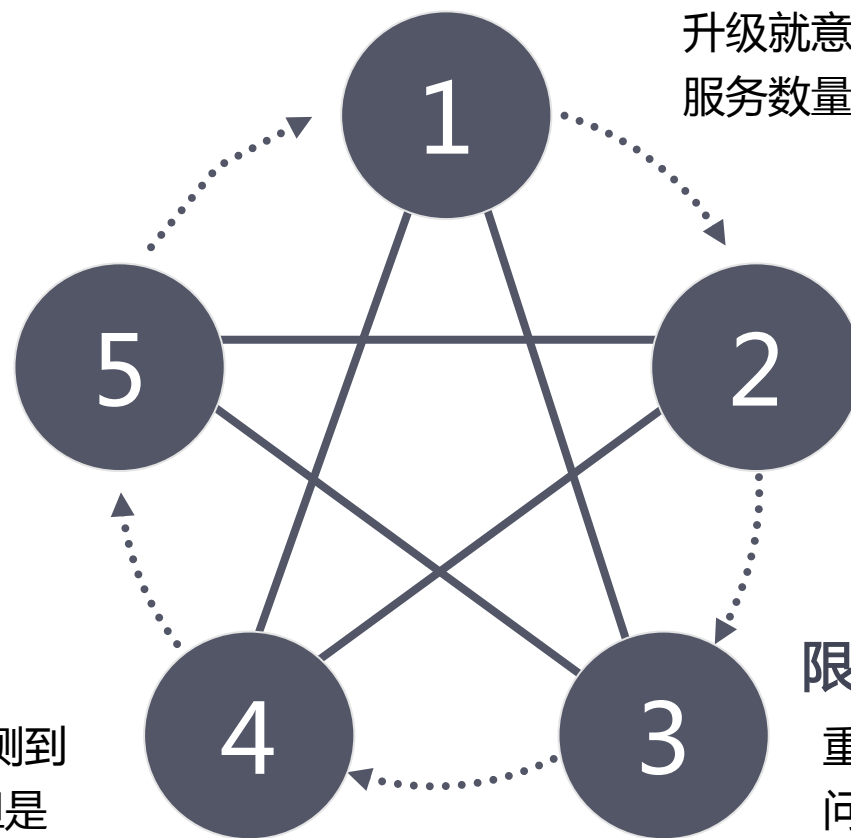
重试虽然可以一定程度上解决问题, 但是重试副作用大, 通常重试次数被严格限制(3次)

## 怎么办?

下游分组升级, 调度算法支持跨组重试, 分组信息持久化在服务注册表中

## 健康检测感知慢

健康检测虽然可以探测到不可用的下游服务, 但是健康检测存在周期性



## 变更分级

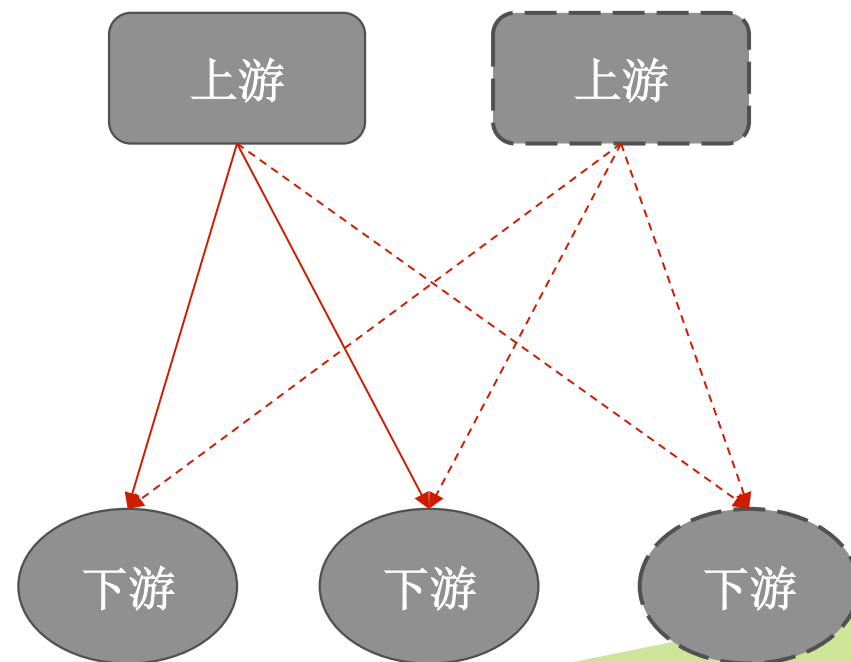
更好的是通过权重来控制下游服务的流量比例

上游分级方式具有随机性, 出错情况损失偏大

基于分布式锁的个数, 控制上游变更的服务

给下游实例打tag, 标记是否被上游可见

变更连接关系高危变更, 一旦错误, 损失很大



## 通知

- zookeeper的通知机制不可靠
- 对注册表依赖过重, 发生局部故障, 影响服务可用性

## 轮询

- 由agent周期性轮询服务注册表
- 引入版本节点, 只有版本变化时, 才获取全量数据
- 增强了运维的可预期性



### 健康检查

- 上游服务探测下游服务健康状态

### 轮询机制

- 使用轮询而非通知机制, 避免通知机制不可靠

### 持久缓存

- 上游持久化缓存下游服务列表, 作为容灾手段

### 墨菲定律

- 历史上发生过多多次zookeeper的局部故障, 比如网络抖动导致大量session超时, 通知消息丢失

### 局部故障

- 不希望由于服务发现系统局部故障而影响服务

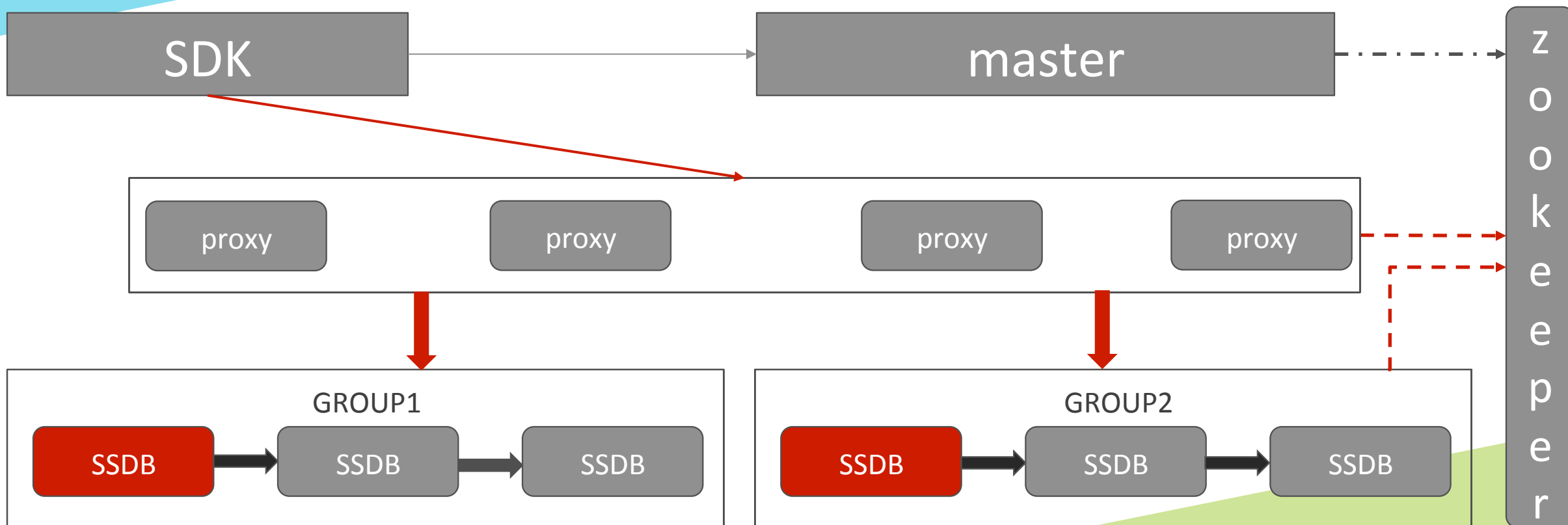






## 应用案例

## 高性能在线服务的分布式KV数据存储服务



## 相关对策

### 原有方案

ssdb向zk注册  
ssdb通过zk选主  
proxy watch zk  
master watch zk  
SDK watch zk  
依赖zk session探活

### 暴露问题

网络抖动session超时  
zk通知机制丢消息  
zk故障服务整体不可用  
平均1~2个月发生故障

### 迁移之后

利用心跳检测和探活  
持久缓存服务列表  
轮询获取服务列表

## 新旧方案对比



## 总结和思考

	consul	Eden
注册表存储	etcd	zookeeper
调用时机	服务自身/consul-template	agent
无损升级	可变服务变更需要额外机制	利用服务分组机制
健康检查	HTTP请求+机器环境	上下游心跳+检查脚本
变更分级	支持下游分级	上游分级+下游分级
变化感知	基于etcd的watch机制	轮询zookeeper
避免单点	依赖稍强	弱依赖

### 总结

- 使用第三方组件注册和注销
- 上游探测下游服务健康状态
- 服务分组实现无损升级
- 连接关系变更一定要分级
- 使用轮询而不是通知
- 以服务注册表不可靠作为假设条件

### 思考

- 引入类似k8s的endpoint机制
- 通过控制流量比例更好的实现分级
- 提升易用性, 成为通用的中间件

GIAC | BEIJING  
Dec.12.16-17

技术架构未来

架構  
ARCHNOTES  
高 可 用 架 构

Thank you!



• 郑然\_百度



• 191895710

