

# 滴滴基于falcon的监控实践

DD-Falcon

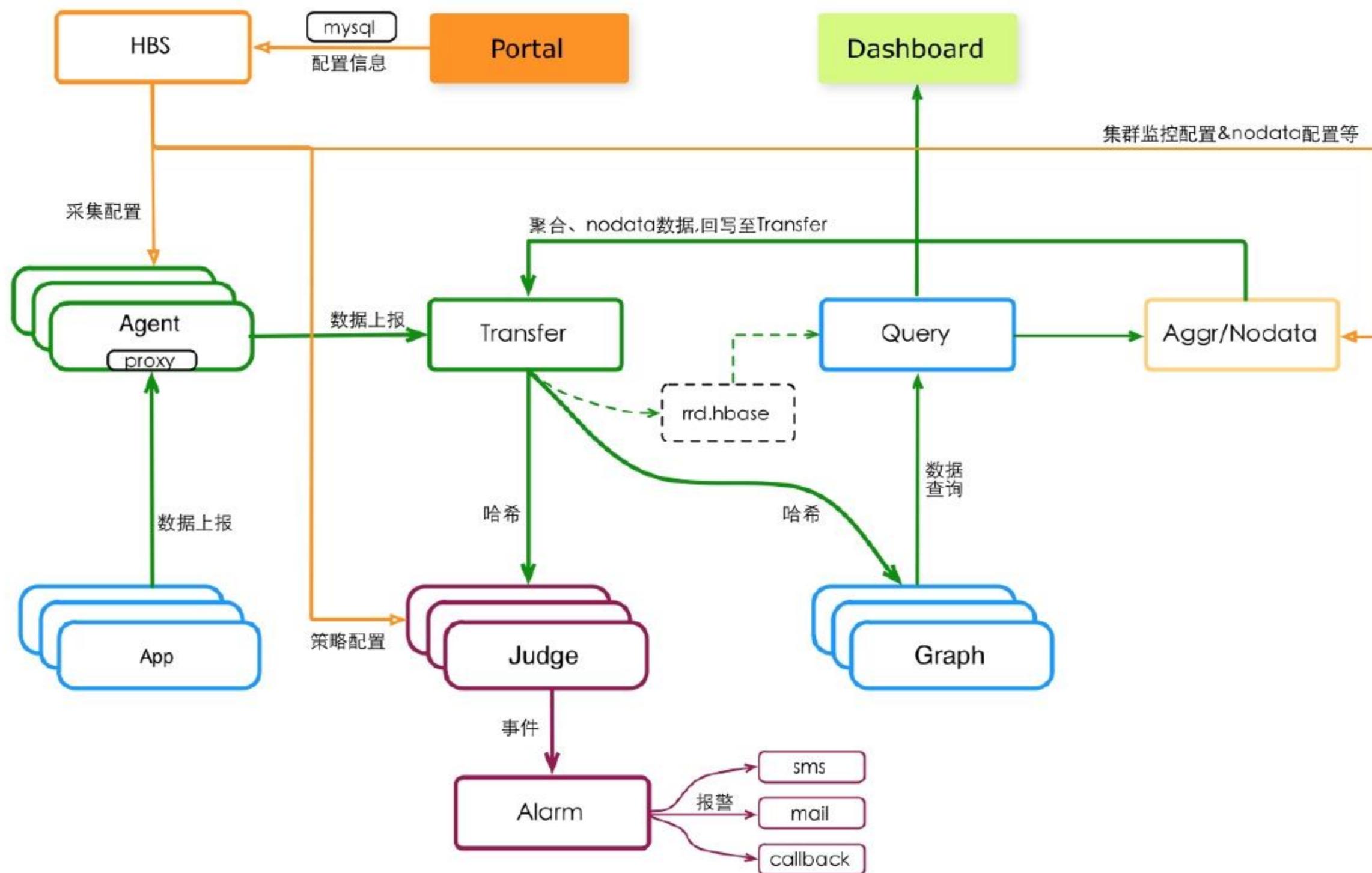
2017-06



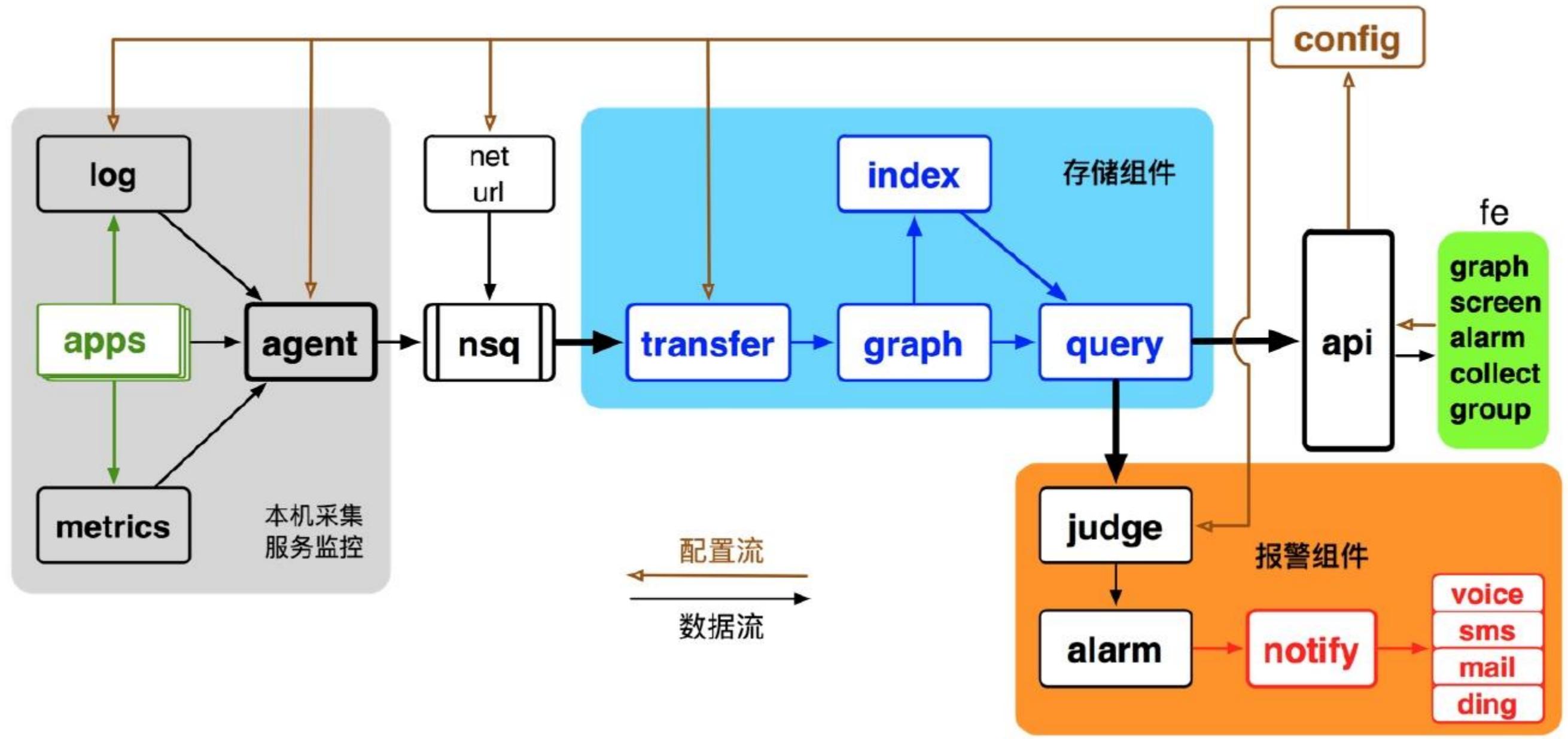
# 主要内容

- 系统架构
- 主要改进
- 已知问题
- 系统规划

# 系统架构: Open-Falcon



# 系统架构: DD-Falcon



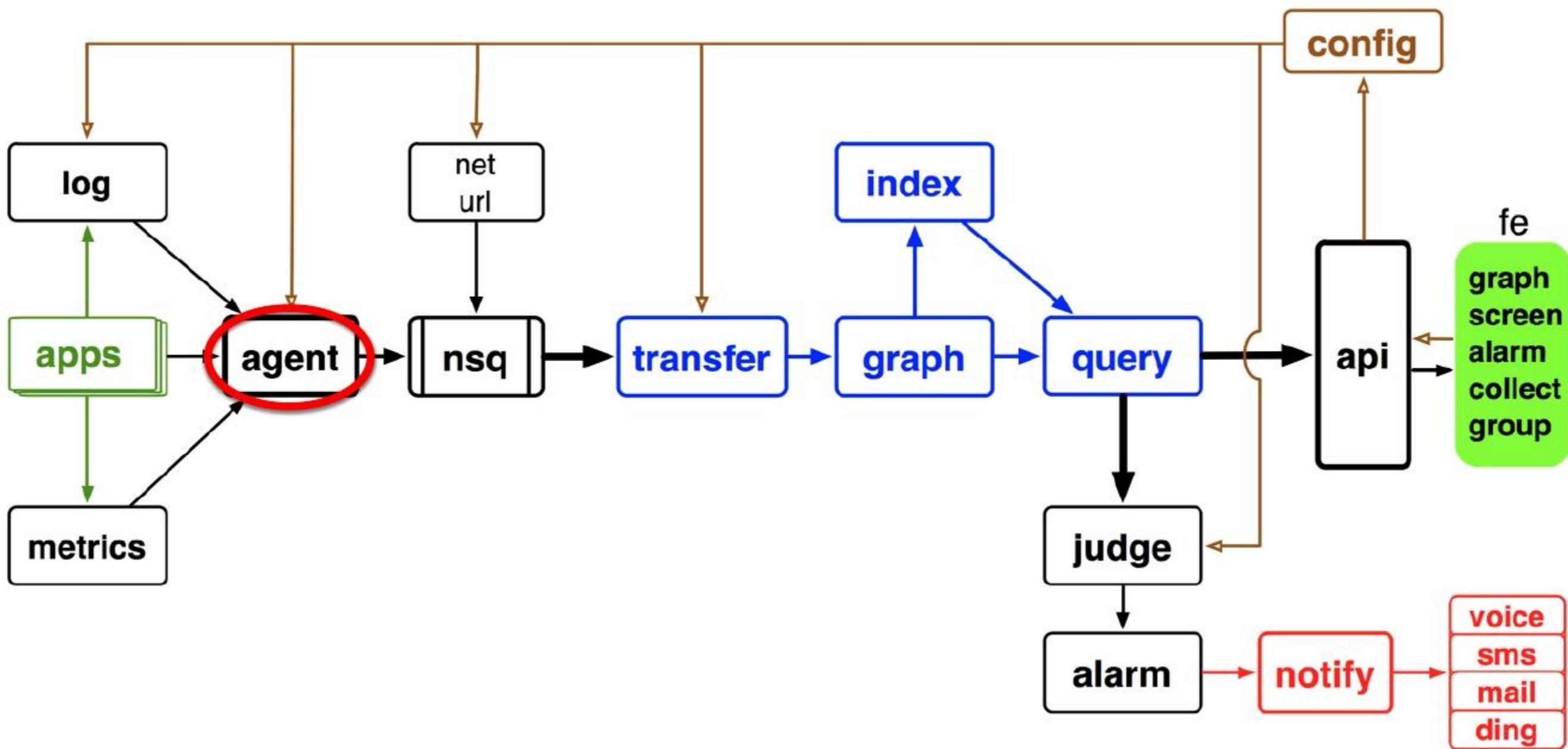
数据流: 服务 → 采集 → 收集 → 清洗 → 存储 → 消费: 报警, 看图, 第三方  
 配置流: 用户 → 配置 → 存储 → 下发 → 生效



# 主要改进

1. 监控数据按服务单元分类
2. 增加垃圾数据清洗
3. 分级索引
4. 精简RRA
5. 巡检大盘支持同环比
6. 重组看图首页
7. 报警数据获取由推变拉
8. 干掉报警模板
9. 重新定义nodata

# 主要改进：数据按照服务单元分类



# 主要改进：数据按照服务单元分类

服务单元 **su =  $\{\text{cluster}\} . \{\text{usn}\}$**

```
{  
  su: “gz01.falcon-graph”,  
  metric: “io.util”,  
  tags: {  
    host: “op-falcon-g.gz01”,  
    device: “nvme0n1”  
  },  
  step: 10,  
  
  “ts”: 1497493910,  
  “value”: 1.0  
}
```

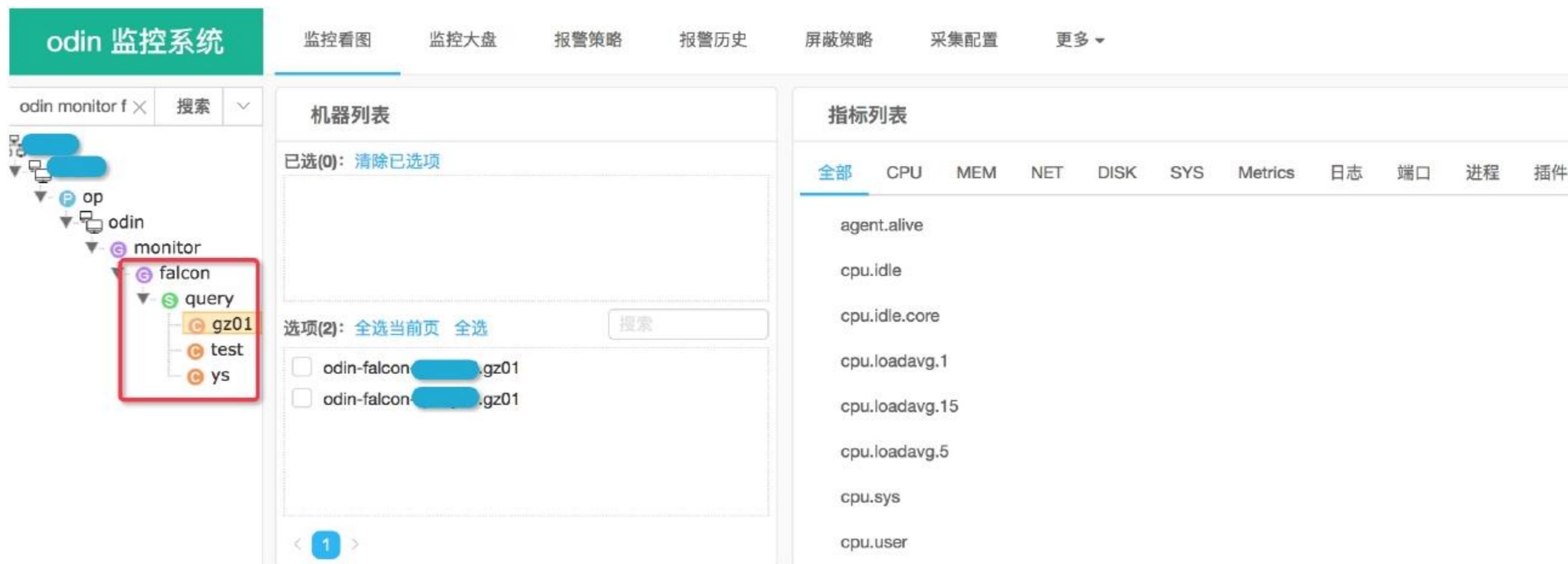
机器指标

```
{  
  su: “gz01.falcon-query”,  
  metric: “rpc.counter”,  
  tags: {  
    host: “op-falcon-q.gz01”,  
    caller: “gz01.falcon-query#query”,  
    callee: “gz01.falcon-graph#series”  
  },  
  step: 10,  
  
  “ts”: 1497493910,  
  “value”: 1.0  
}
```

服务间rpc调用质量

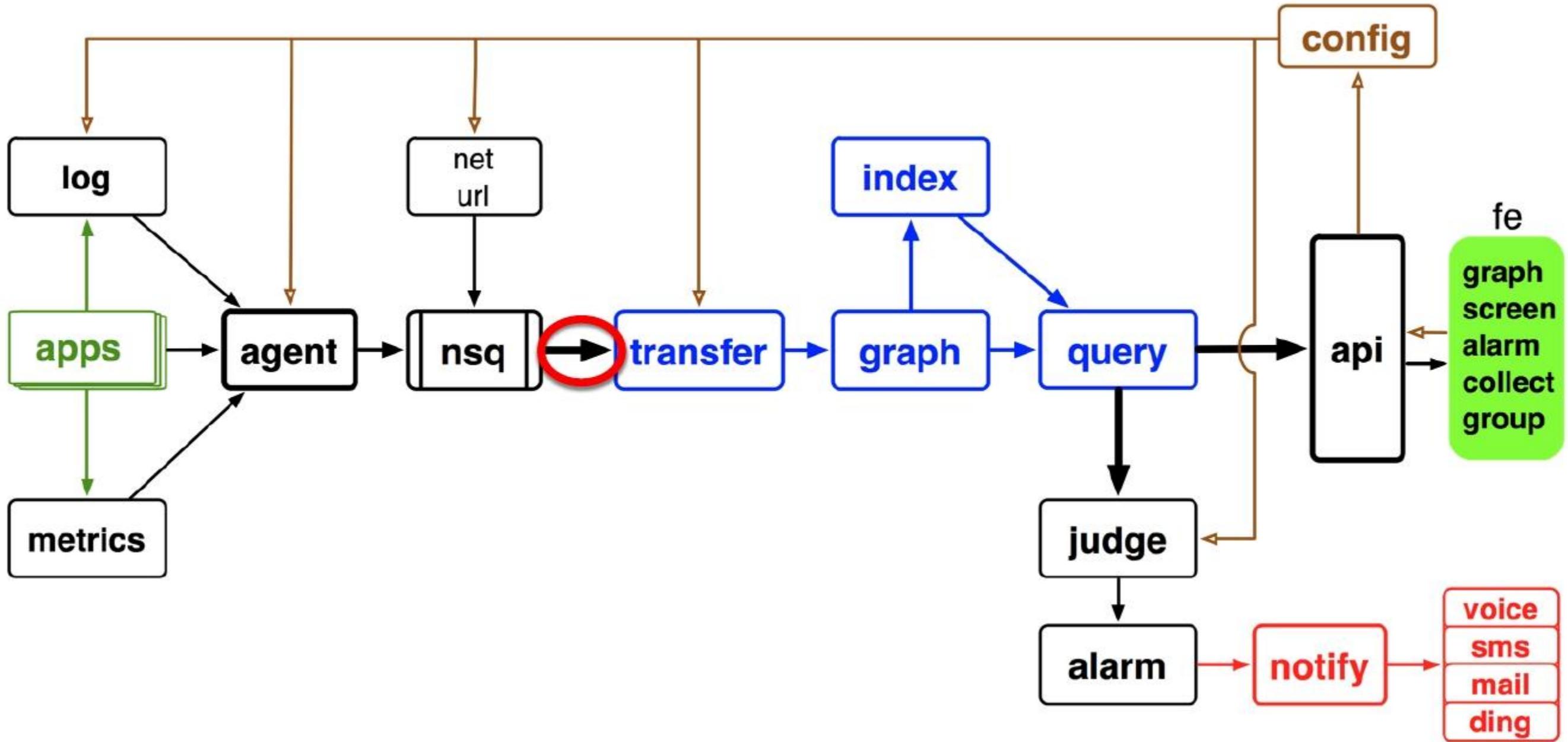
# 主要改进：数据按照服务单元分类

su与服务树节点一一对应, 查看某个服务的监控会很方便



The screenshot displays the Odin Monitoring System interface. On the left, a service tree is shown with a red box highlighting the 'falcon' service and its sub-nodes 'query', 'gz01', 'test', and 'ys'. The main area is divided into two panels: '机器列表' (Machine List) and '指标列表' (Metrics List). The '机器列表' panel shows a search bar and a list of machines, with two entries for 'odin-falcon' and 'gz01'. The '指标列表' panel shows a list of metrics under the '全部' (All) tab, including 'agent.alive', 'cpu.idle', 'cpu.idle.core', 'cpu.loadavg.1', 'cpu.loadavg.15', 'cpu.loadavg.5', 'cpu.sys', and 'cpu.user'.

# 主要改进：垃圾数据清洗





# 主要改进：垃圾数据清洗

```
{  
  su: "gz01.falcon-query",  
  metric: "rpc.counter",  
  tags: {  
    host: "op-falcon-q.gz01",  
    trace: "0ed9c487 ...",  
  },  
}
```

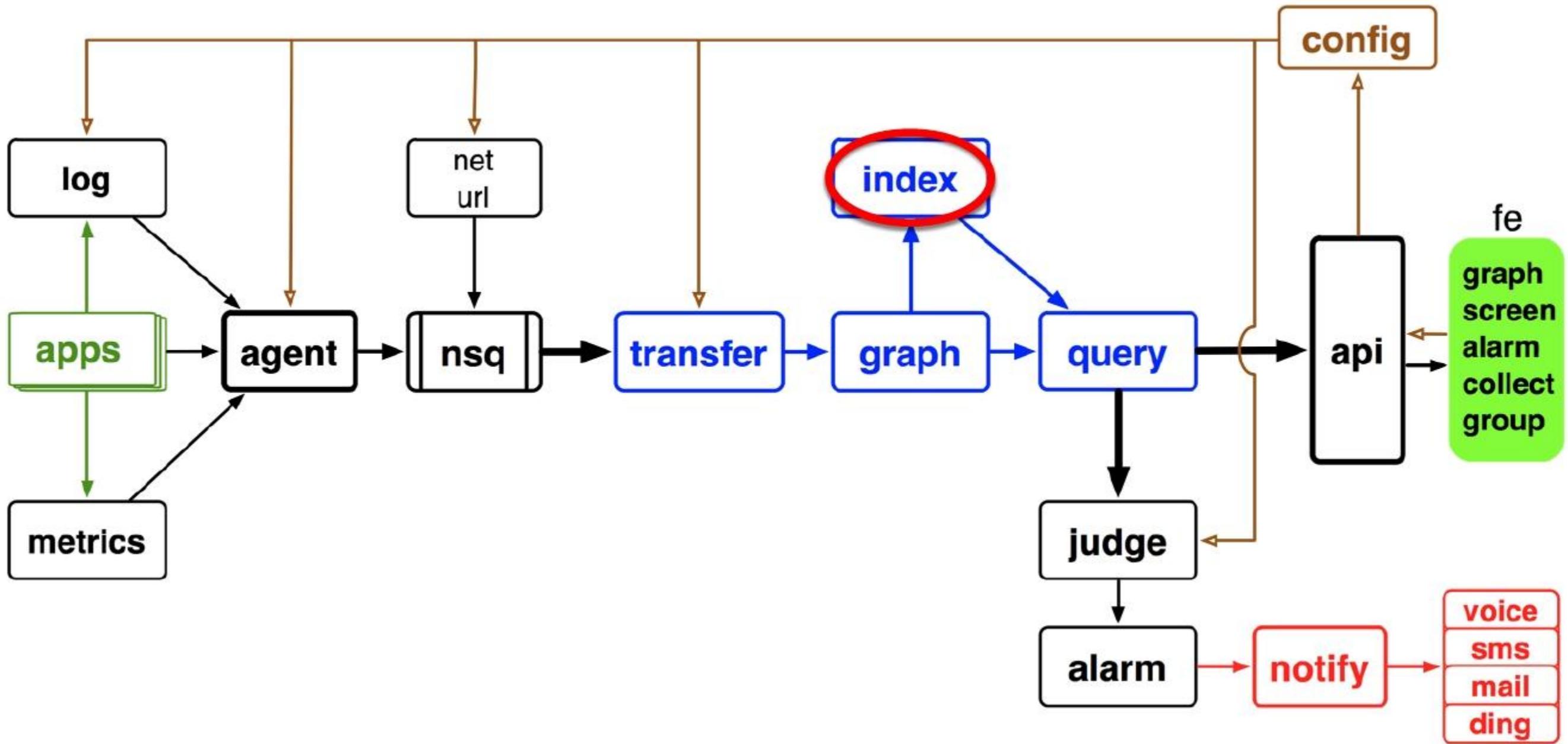
清洗规则:

```
su    equal (gz01.falcon-query) &&  
metric equal (rpc.counter) &&  
tagk  equal (trace)
```

1. 清洗维度: 服务单元su, 指标metric, tagk, tagv, metric/tagk
2. 清洗方式: 字符串 相等, 前缀, 后缀, 包含

**95%**的清洗规则, 是通过 **tagv**前缀匹配 实现的

# 主要改进：分级索引

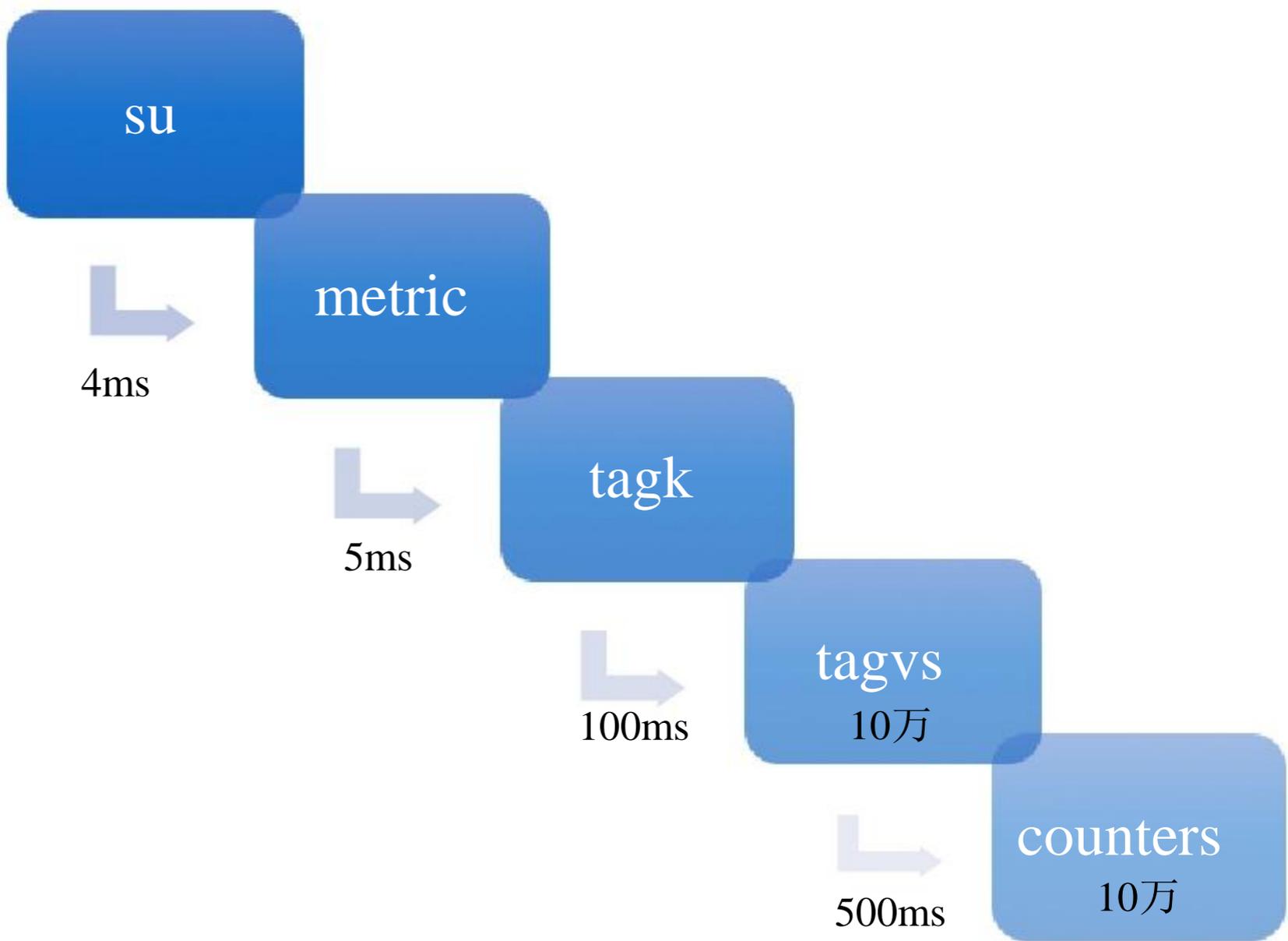




# 主要改进：分级索引

根据滴滴用户习惯, 实现多级索引结构, 读取数据更灵活

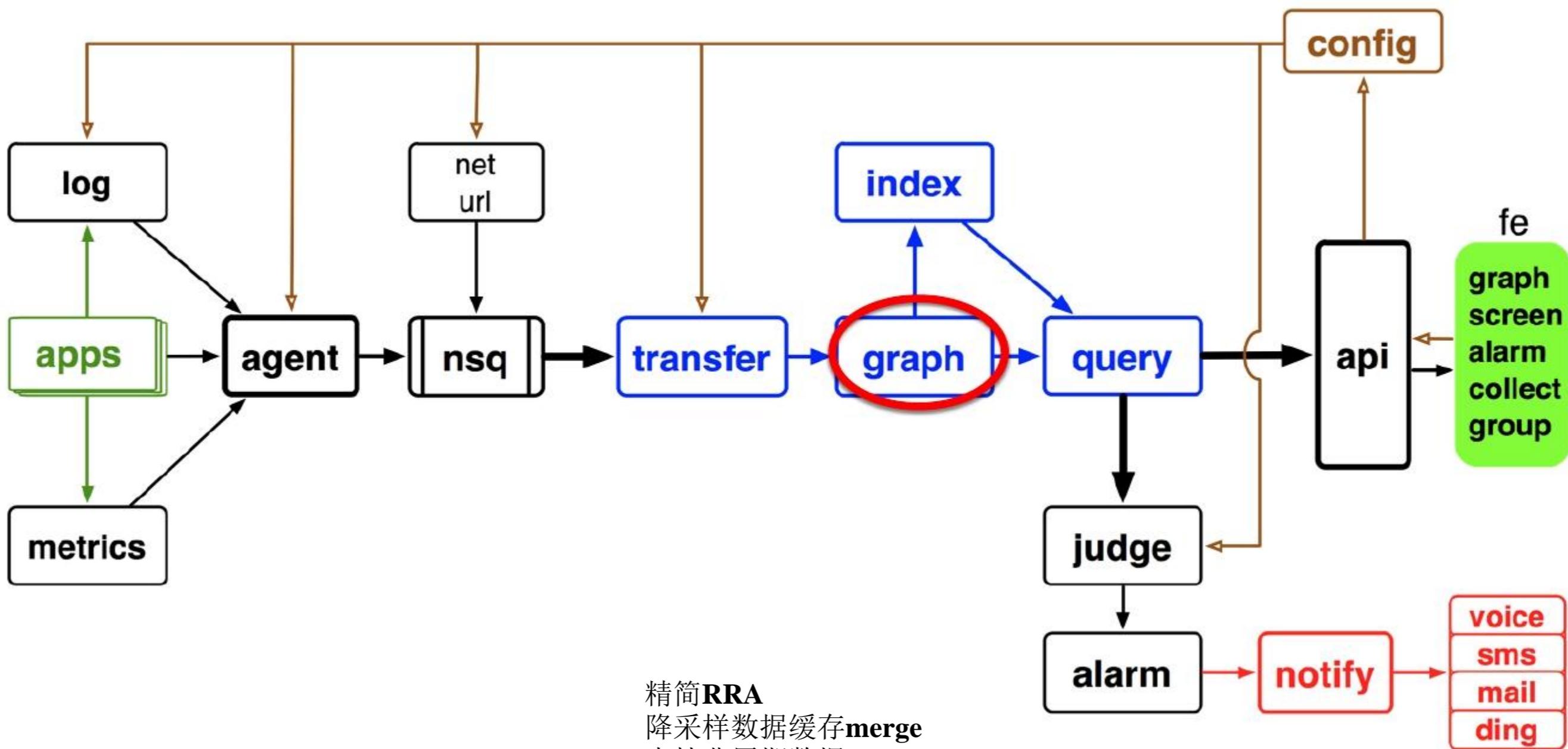
```
{  
  su: "gz01.f-gr",  
  metric: "io.util",  
  tags: {  
    host: "op-f-gr.gz01",  
    device: "nvme0n1"  
  },  
  step: 10,  
  
  "ts": 1497493910,  
  "value": 1.0  
}
```



1000万指标: 构建耗时30s, 消耗内存2GB

1亿 指标: 构建耗时5min, 消耗内存17GB

# 主要改进: 精简RRA





# 主要改进: 精简RRA

```
// 设置各种归档策略
// 1分钟一个点存 12小时
c.RRA("AVERAGE", 0.5, 1, RRA1PointCnt)

// 5m一个点存2d
c.RRA("AVERAGE", 0.5, 5, RRA5PointCnt)
c.RRA("MAX", 0.5, 5, RRA5PointCnt)
c.RRA("MIN", 0.5, 5, RRA5PointCnt)

// 20m一个点存7d
c.RRA("AVERAGE", 0.5, 20, RRA20PointCnt)
c.RRA("MAX", 0.5, 20, RRA20PointCnt)
c.RRA("MIN", 0.5, 20, RRA20PointCnt)

// 3小时一个点存3个月
c.RRA("AVERAGE", 0.5, 180, RRA180PointCnt)
c.RRA("MAX", 0.5, 180, RRA180PointCnt)
c.RRA("MIN", 0.5, 180, RRA180PointCnt)

// 12小时一个点存1year
c.RRA("AVERAGE", 0.5, 720, RRA720PointCnt)
c.RRA("MAX", 0.5, 720, RRA720PointCnt)
c.RRA("MIN", 0.5, 720, RRA720PointCnt)
```



```
// 设置各种归档策略
// 10s 一个点存2小时
c.RRA("AVERAGE", 0, 1, RRA1PointCnt)

// 1min 一个点存8天
c.RRA("AVERAGE", 0, 6, RRA6PointCnt)

// 30min 一个点存1月
c.RRA("AVERAGE", 0, 180, RRA180PointCnt)

// 6h 一个点存1年
c.RRA("AVERAGE", 0, 1080, RRA1080PointCnt)
```

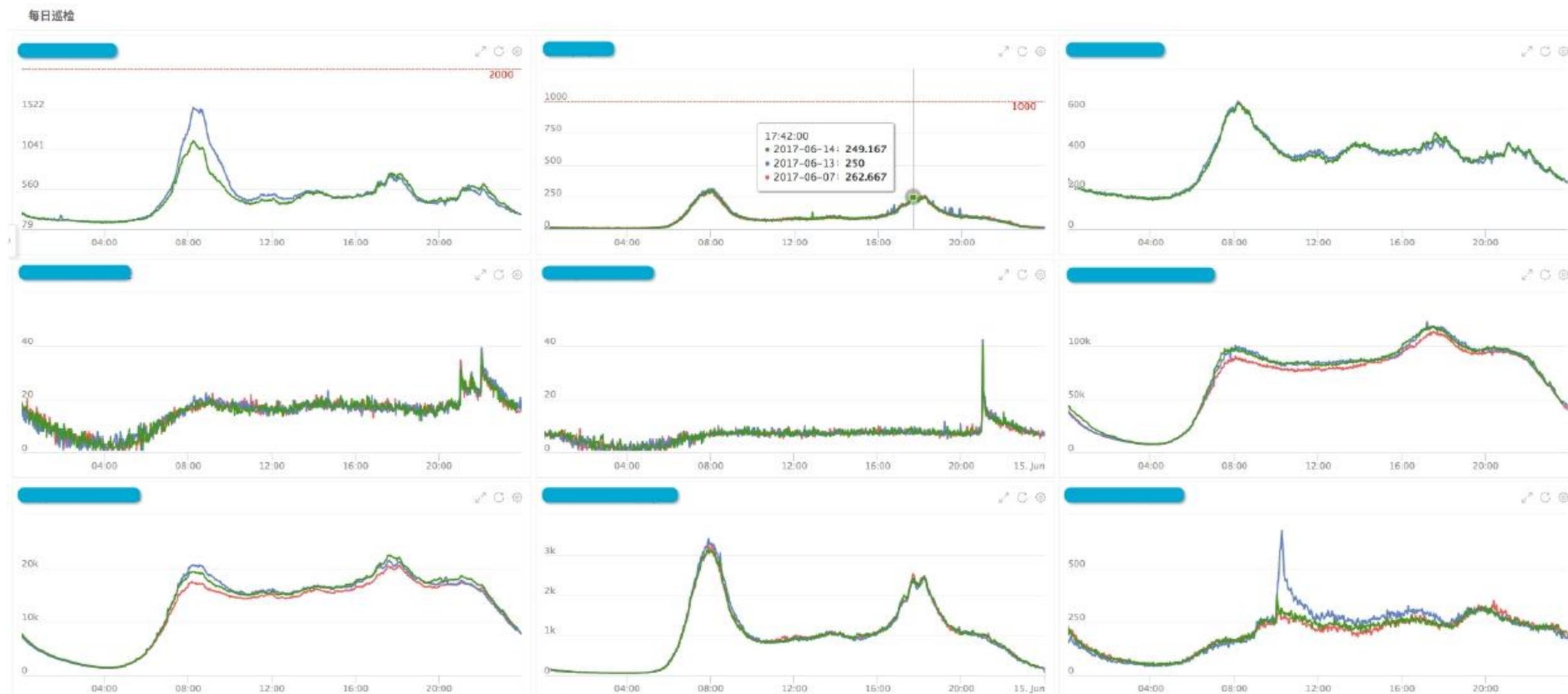
干掉 MAX、MIN, 只支持 AVG 的归档

存储容量消耗降低60%, io消耗降低30%

1分钟的归档数据存8天, 为同环比提供支持



# 主要改进：大盘支持同环比看图



典型的每日巡检大盘。60%+的巡检大盘, 都是同环比

# 主要改进：同环比配置

只支持1天和7天的同环比, 这是由业务的周期特点决定

\* 节点: collect.gz01.query.falcon.monitor.odin.op [redacted] ×

\* 指标: rpc.counter

聚合: 无 > 环比: 1天 ×

时间: 1小时

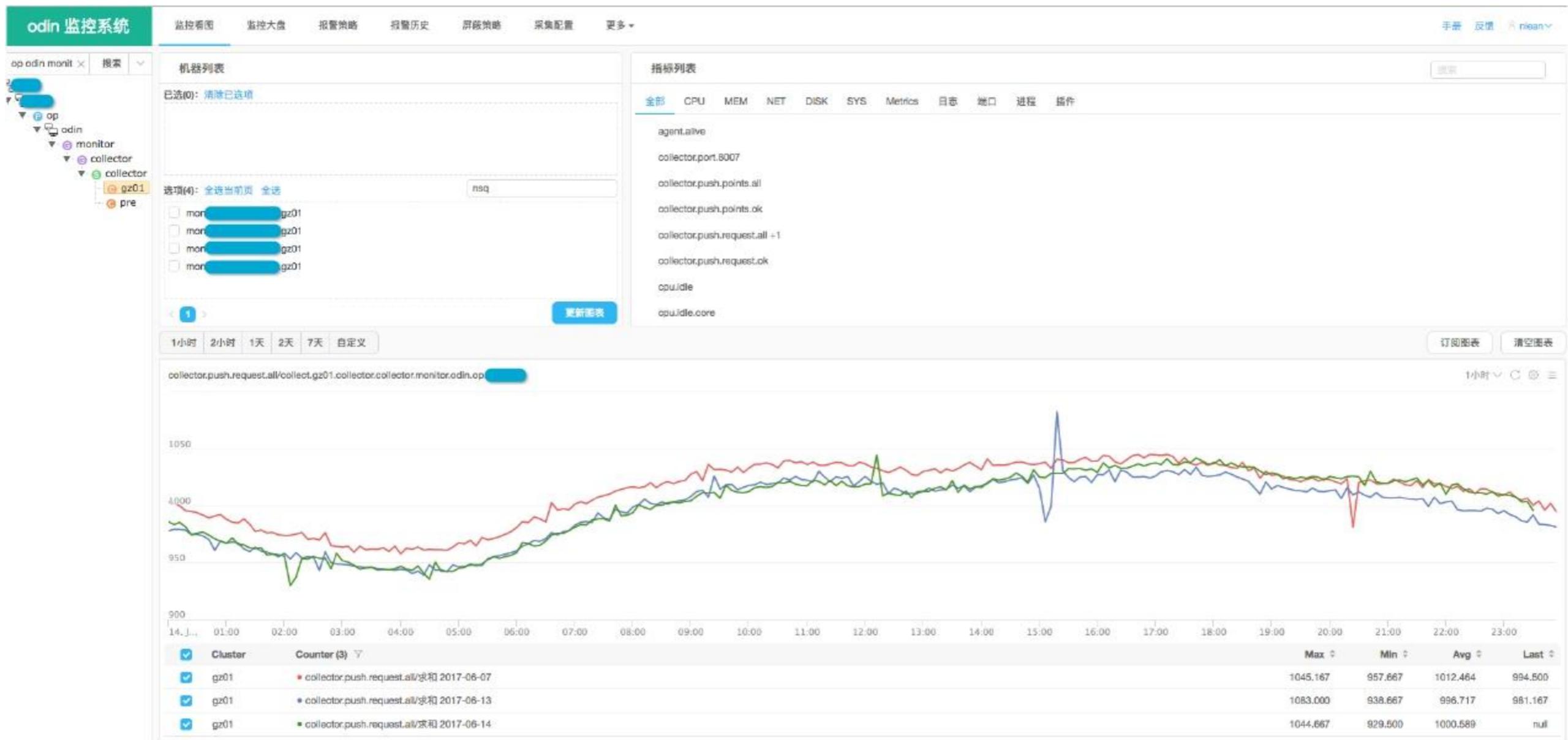
\* host: odin-falcon-[redacted].gz01

\* callee: odin-graph

\* caller: odin-query, odin-query-batch

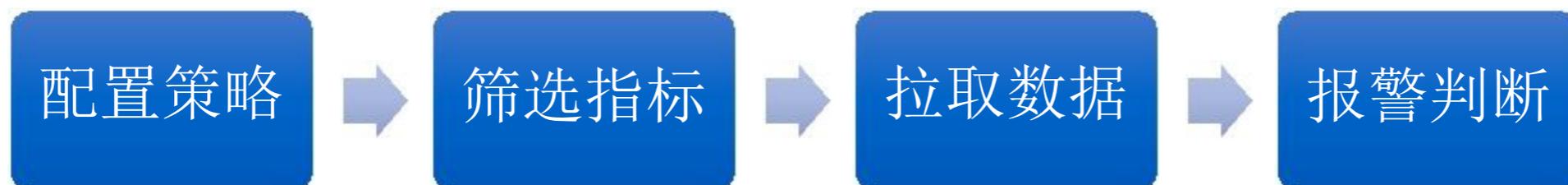
cluster	counter (2 x 2)	step
gz01	callee=odin-graph,caller=odin-query,host=odin-falcon-[redacted].gz01	10
gz01	callee=odin-graph,caller=odin-query-batch,host=odin-falcon-[redacted].gz01	10

# 主要改进: 重组看图首页



看图步骤: 服务单元 → 节点 → 机器 → 指标分组 → 看图 → 订阅大盘

# 主要改进：报警数据获取由推变拉



拉数据更灵活,可以实现各种判断条件: 多指标组合,同环比,集群

## 1. 多指标组合条件



The screenshot shows a configuration interface for a monitoring rule. It consists of two condition blocks connected by an 'AND' operator.

**Condition 1:**  
Metric: `cpu.idle`  
Trigger: 连续发生 (all)  
n: 6, v:  $\geq$ , 90  
Preview: `cpu.idle` 最近 6 个周期每个值都  $\geq 90$

**Condition 2:**  
Metric: `mem.used.percent`  
Trigger: 发生次数 (happen)  
n: 12, m: 6, v:  $\geq$ , 80  
Preview: `mem.used.percent` 最近 12 个周期内 6 次值  $\geq 80$

## 2. 同环比报警



The screenshot shows a configuration interface for a monitoring rule with a percentage change condition.

**Condition:**  
Metric: `nscounter.nscnt`  
Trigger: 同比变化率 (c\_avg\_rate\_abs)  
n: 6, m: 7, v:  $>$ , 10  
Preview: `nscounter.nscnt` 绝对值(最近 6 个周期平均值相对 7 天前同周期平均值变化)  $> 10\%$

# 主要改进：干掉报警模板

增加了策略的管理成本, 但大大降低了用户的学习成本

功能场景	旧实现方式	新实现方式	对新方式的评价
添加报警	新建模板&将模板绑定到节点	在节点上新建策略	<ul style="list-style-type: none"><li>易于理解: 无需模板这一概念</li></ul>
排除特例	模板覆盖&报警组留空	排除节点	<ul style="list-style-type: none"><li>易于理解: 排除更直观</li><li>简化操作: 一步到位</li></ul>
修改子节点的接收组	模板覆盖&更新报警组	排除节点&新建策略	<ul style="list-style-type: none"><li>增加操作: 既要修改老策略, 又要建新策略</li></ul>
复用配置	模板继承	策略克隆	<ul style="list-style-type: none"><li>易于理解: 克隆更直接</li><li>管理困难: 更新配置时需遍历所有同类策略</li></ul>



# 主要改进：干掉报警模板

odin 监控系统

监控看板 监控大盘 报警策略 报警历史 屏蔽策略 采集配置 更多

报警策略管理 (gz01.collector.collector.monitor.odin)

新增策略 批量删除 级别: ALL 接收组: 为空表示全部 操作者: 为空表示全部 搜索

级别	策略名称	节点	接收组	最后操作者	最后修改时间	操作
P1	collector服务端口死掉	collector.collector.monitor.odin	odin-monitor		2017-05-05 17:49:40	详情   克隆
P3	发送数据的请求量大于5000	collector.collector.monitor.odin	odin-monitor		2017-05-05 17:51:28	详情   克隆
P3	io.util大于90%	op	op-all		2017-06-07 14:56:36	详情   克隆
P3	cpu.idle小于10%	op	op-all		2017-06-07 14:56:48	详情   克隆
P3	机器死机	op	op-all		2017-06-14 12:05:17	详情   克隆
P3	网卡入流量大于90%	op	op-all		2017-06-07 14:57:14	详情   克隆
P3	mem.used.percent大于90%	op	op-all		2017-06-07 14:57:30	详情   克隆
P3	网卡出流量大于90%	op	op-all		2017-06-07 14:57:51	详情   克隆
P3	磁盘空间大于95%	op	op-all		2017-06-14 10:02:35	详情   克隆
P3	磁盘inode使用率大于95%	op	op-all		2017-06-07 19:53:36	详情   克隆
P3	系统触发OOM	op	op-all		2017-06-07 14:59:33	详情   克隆
P3	机器coredump	op	op-all		2017-06-13 12:10:27	详情   克隆



# 主要改进：重新定义nodata

重新定义业务场景：

1. 正常上报的数据突然中断了, 才需要nodata报警
2. 从来没上报过的数据, 没有必要nodata报警

主要收益是简化配置：

1. 支持按照tag过滤指标, 无需指定具体指标
2. 统一了nodata报警的配置方式, 与普通报警配置无差异

\* 触发条件：

常用

service.alive

数据上报中断 (nodata)

n: 600

预览: service.alive 最近 600 秒无数据上报



# 主要改进 - 总结

1. 监控数据按服务单元分类
2. 增加垃圾数据清洗
3. 分级索引
4. 精简RRA
5. 巡检大盘支持同环比
6. 重组看图首页
7. 报警数据获取由推变拉
8. 干掉报警模板
9. 重新定义nodata



# 已知问题

## 1. 非周期的数据

- ① 报警延时风险
- ② 断点, 环比看图不易发现问题
- ③ 历史数据严重有损

## 2. 打通非时间序列化的系统

- ① trace



# 系统规划

## 1. 精确的报警定位能力

- ① 低成本的阈值预测
- ② 集群聚合能力
- ③ 服务间报警关联

## 2. 时间序列化数据平台

- ① 个性化的看图解决方案

## 3. 打通非时间序列化的系统



谢谢！！