# 弹性工程–构建弹性系统

曾绿麟@美团点评境内度假 20170325

# 个人简介

- 2010年 华中科技大学硕士毕业
- 2010~2013年 人人网 基础架构类开发
- 2013~now 美团点评周边游C端技术负责人

# 大纲

- 引子
- 模式篇
- 工具篇
- 最后的建议

# 引子

可扩展性

可用性

弹性?从来没听过...

可靠性

可维护性

# Resilience

- The ability of a system to handle unexpected situations
  - without the user noticing it (best case)
  - with a graceful degradation of service (worst case)

# 弹性系统

- 面对各种不确定场景时（如基础存储设施故障，恶意爬虫袭击，依赖下游服务故障，网络超时，专线中断等等）都能够存活并且具备一定的自愈能力的系统。

# 为什么要构建弹性系统?

- 一切都为了可用性
- 99.9%到99.99%

| Level of Availability | Percent of Uptime | Downtime per Year | Downtime per Day |
|---|---|---|---|
| 1 Nine | 90% | 36.5 days | 2.4 hrs. |
| 2 Nines | 99% | 3.65 days | 14 min. |
| 3 Nines | 99.9% | 8.76 hrs. | 86 sec. |
| 4 Nines | 99.99% | 52.6 min. | 8.6 sec. |
| 5 Nines | 99.999% | 5.25 min. | .86 sec. |
| 6 Nines | 99.9999% | 31.5 sec. | 8.6 msec |

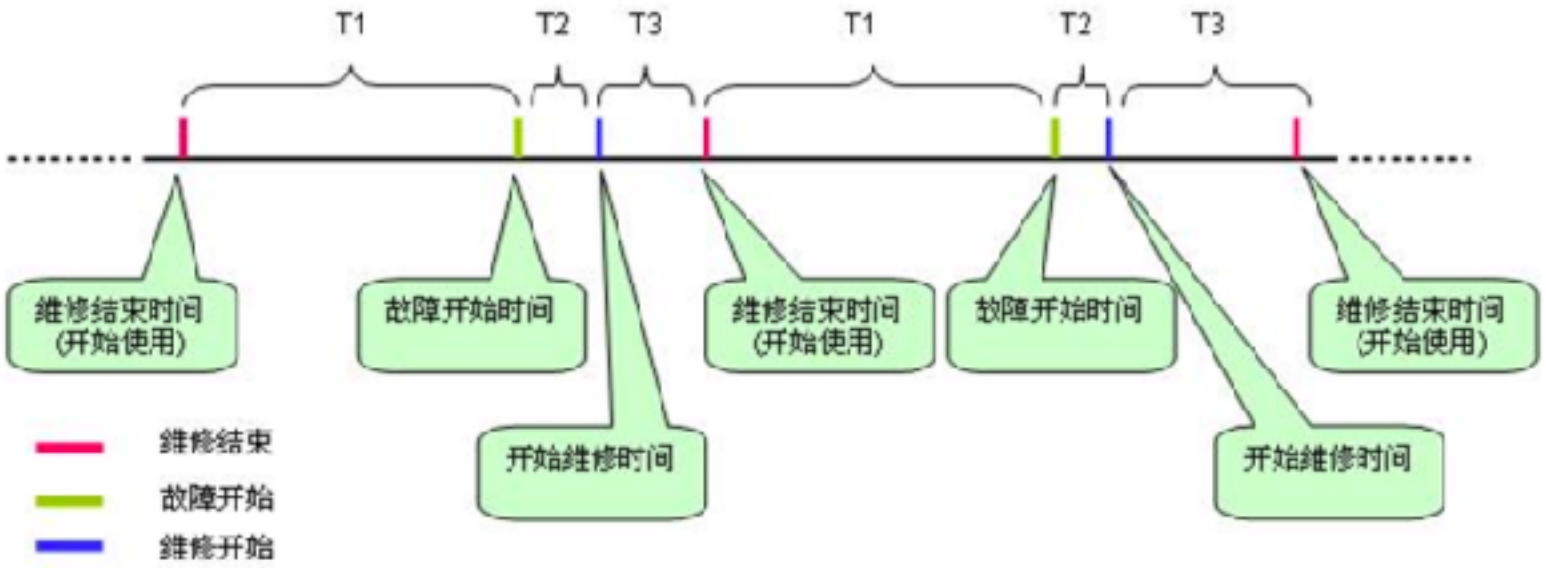
$$\text{Availability} := \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

MTTF: Mean Time To Failure

MTTR: Mean Time To Recovery

# MTTF与MTTR

- MTTF (Mean Time To Failure，平均无故障时间)，指系统无故障运行的平均时间，取所有从系统开始正常运行到发生故障之间的时间段的平均值。 MTTF =∑T1/ N
- MTTR (Mean Time To Repair，平均修复时间)，指系统从发生故障到维修结束之间的时间段的平均值。MTTR =∑(T2+T3)/ N

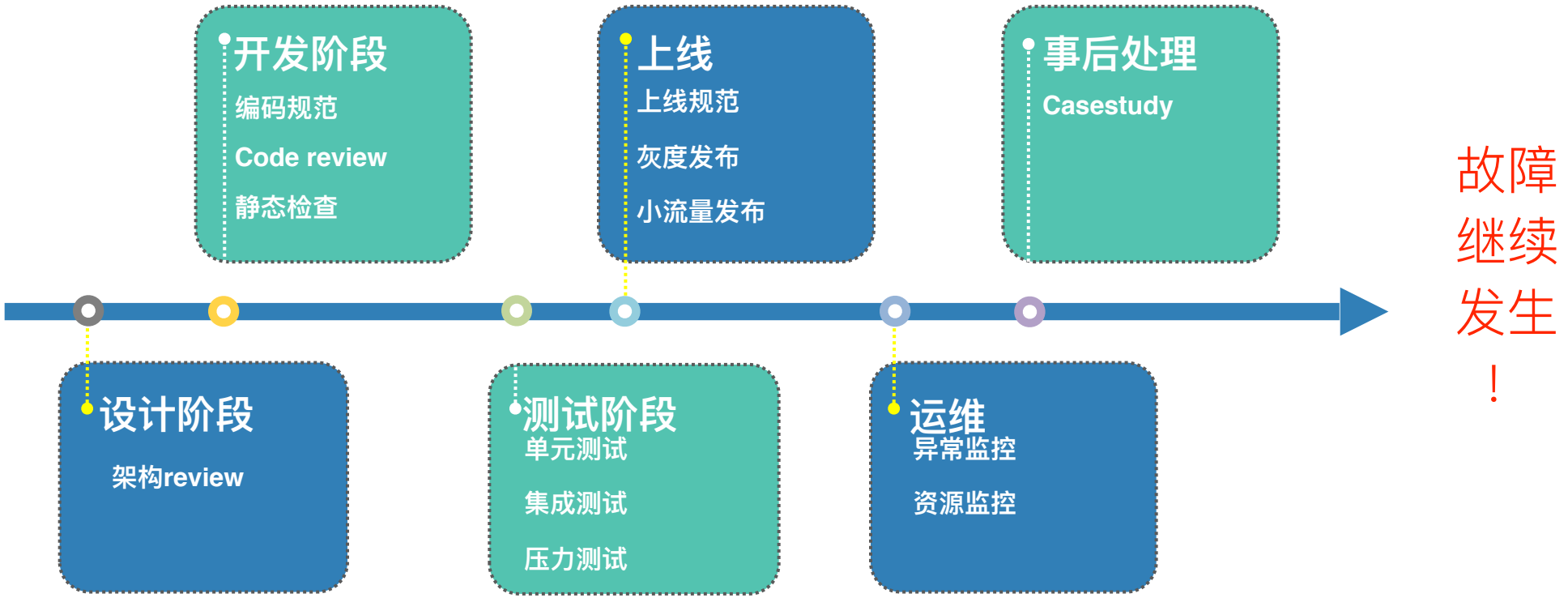


# Traditional stability approach

$$\text{Availability} := \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Maximize MTTF

# 如何提高MTTF?




故障继续发生！

我们在开发的各个阶段都建立了相应的规范，通过SOP保平安！

# 分布式8大谬论

- 网络是可靠的
- 网络延时为零
- 带宽是无限的
- 网络是安全的
- 网络拓扑不会改变
- 肯定至少有一个管理员（在值班）
- 传输开销为零
- 网络是同质的

规模化分布式服务的场景下，

故障是常态的，不可预测的！

Do not try to avoid failures!

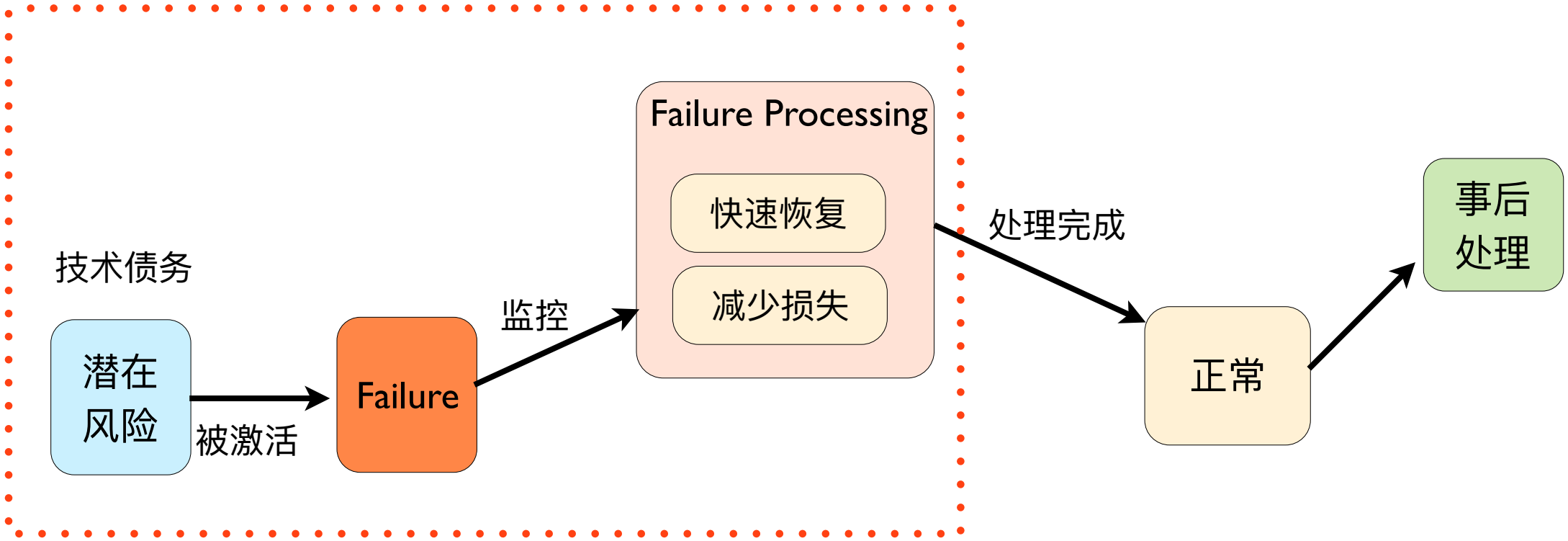
Embrace them!

# Resilience approach

$$\text{Availability} := \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

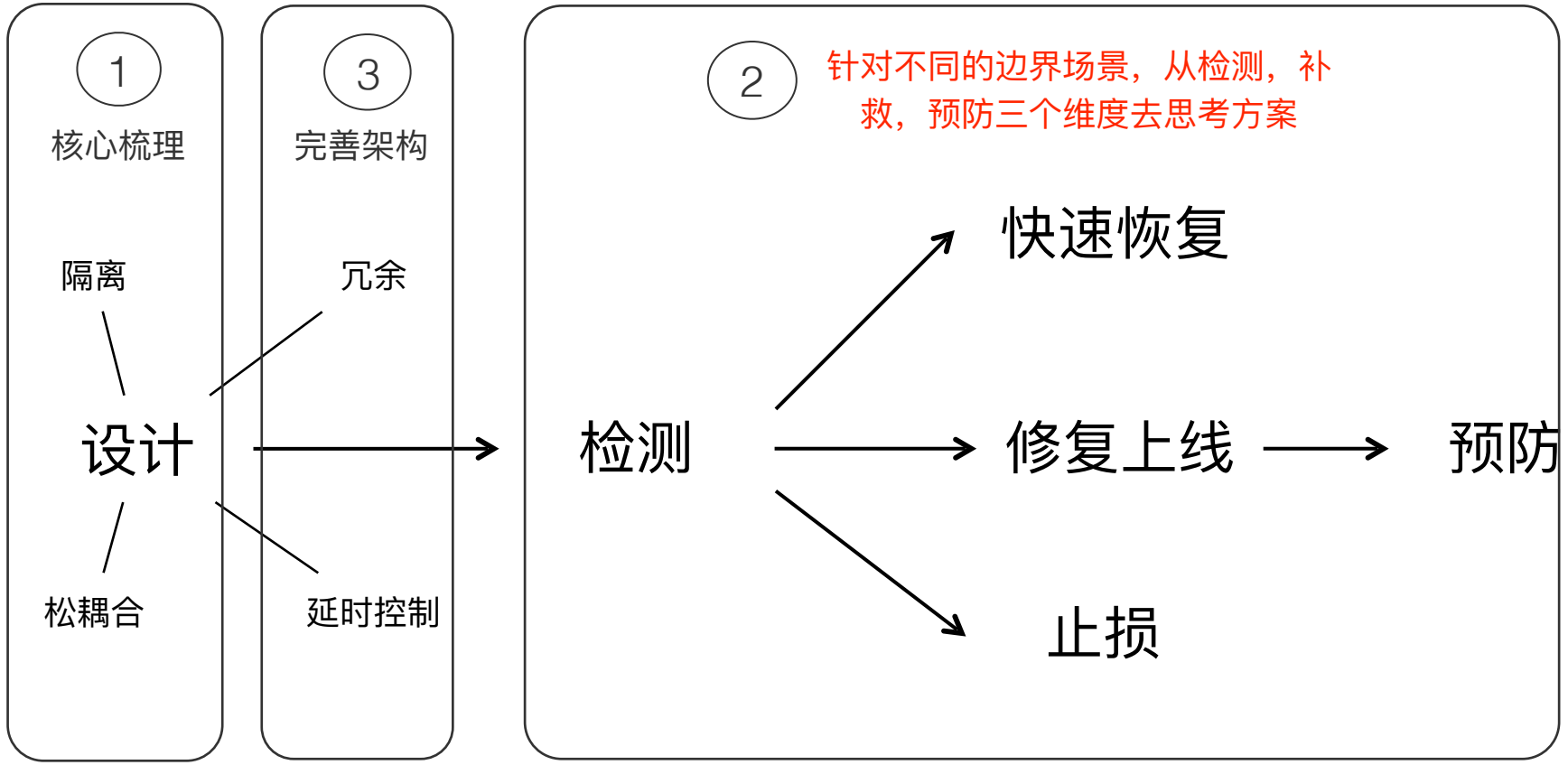
Minimize MTTR

# 故障生命周期




缩短MTTR：提前识别/规避风险+缩短故障发现时间+缩短故障处理时间

# 弹性工程–故障驱动的架构方法




Ongoing

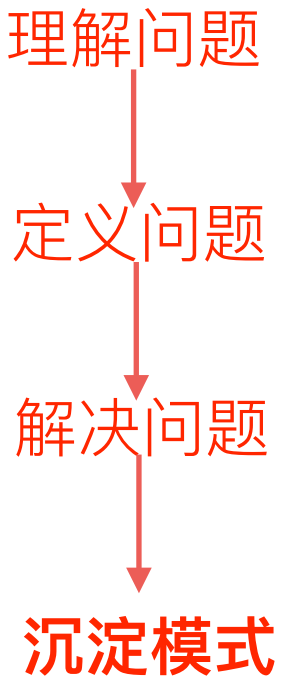
系统分解，微服务化

# 弹性工程–故障驱动的架构方法

- 来自于航天工程领域的一套工程方法，逐步延生到互联网领域
- 缩短MTTR：提前识别/规避风险+缩短故障发现时间+缩短故障处理时间
- 通过提出边界场景（失败/风险/意外事件）问题，然后从检测，补救，预防几个维度去思考解决方案，最终反哺到系统设计开发与流程改善上，倒逼架构和流程SOP改进，再结合预案演练达到扼杀故障和缩短故障处理时间的目的

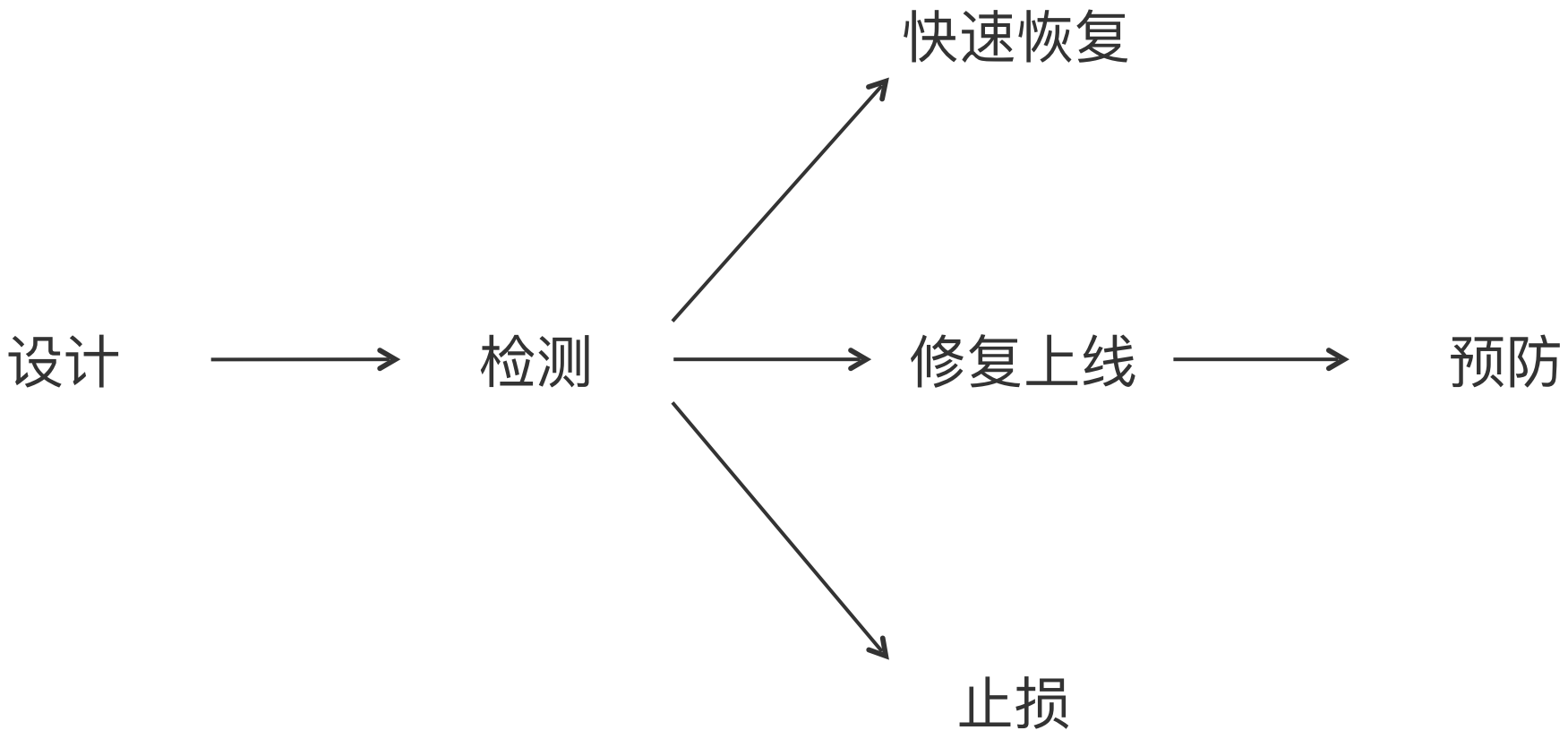
# 常见的边界问题

- 上线出了问题怎么办？
- 机器/机房/专线出了问题怎么办？
- 依赖服务出现异常怎么办？
- 依赖服务出现超时怎么办？
- DB出现慢查询怎么办？
- 基础设施故障（medis/tair/zk/es）出现问题怎么办？
- 异常流量（过载/爬虫/攻击）来了怎么办？
- 监控报警失效怎么办？



# 模式篇

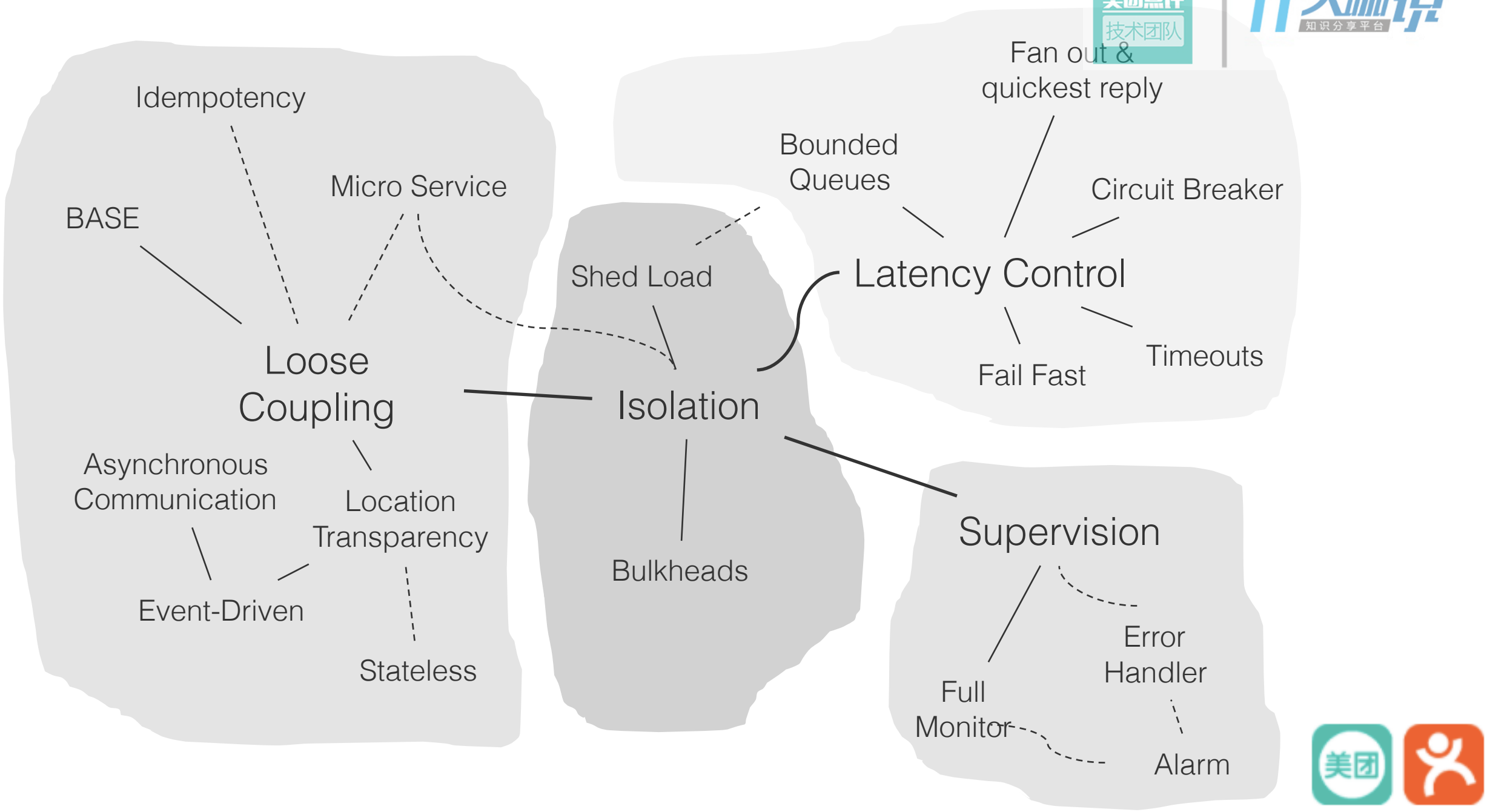
事前　　事中　　事后



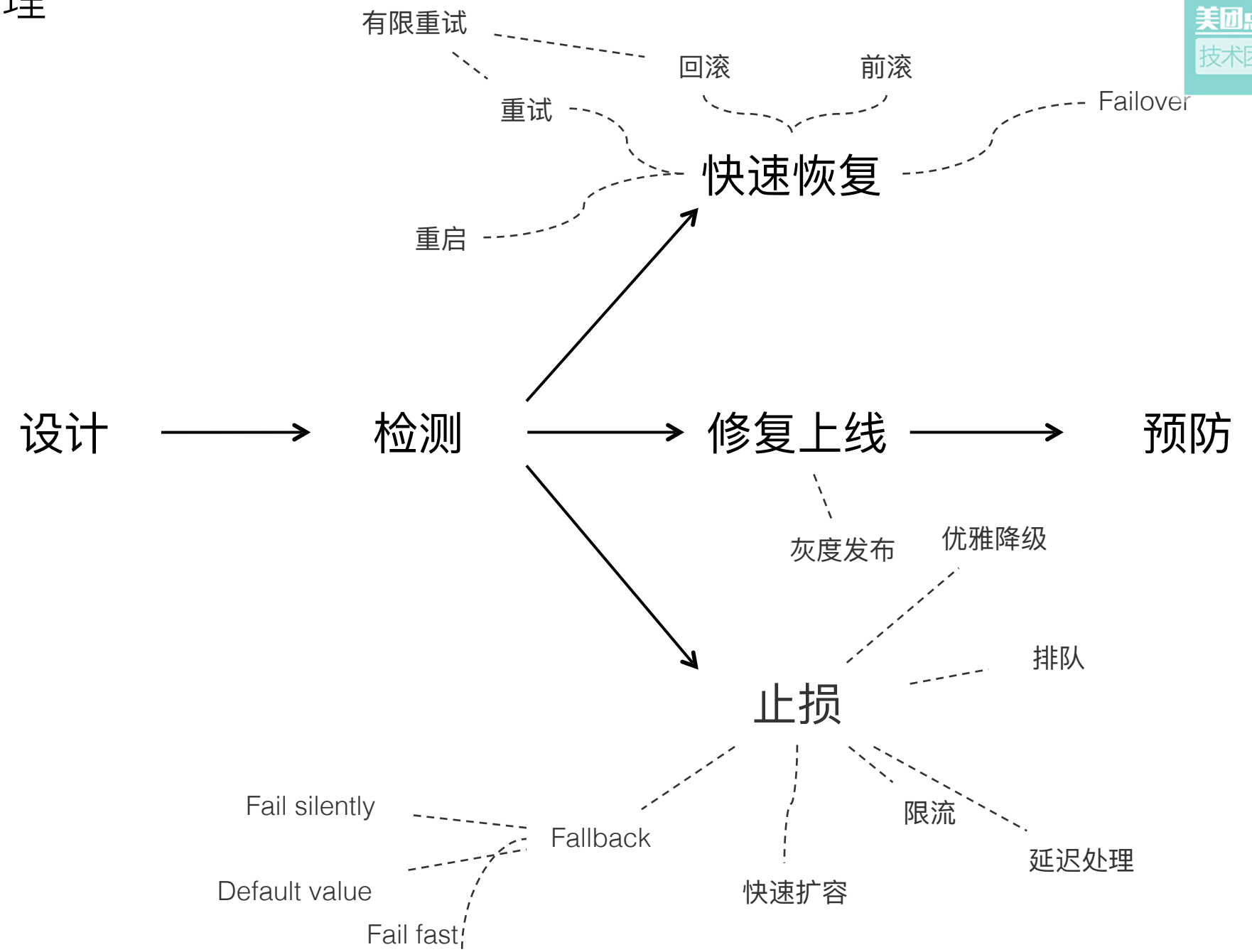

提前识别/规避风险　　缩短故障发现时间　　缩短故障处理时间　　复盘识别/规避风险

# 事前阶段

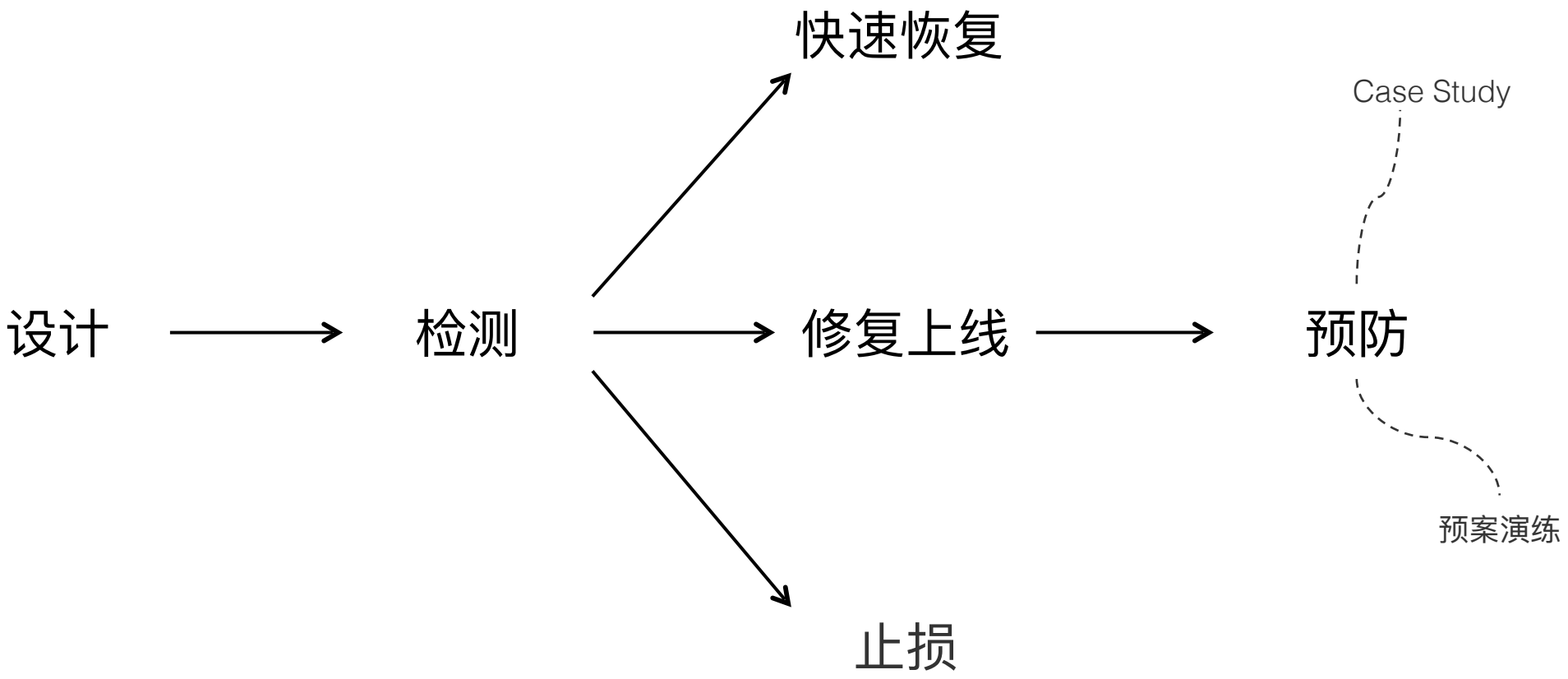
- 设计阶段
  - 隔离：过载保护/舱壁/微服务
  - 延时控制：超时/熔断/有界队列等
  - 松耦合：柔性可用/幂等/事件驱动(异步)/无状态/位置透明等
  - 冗余：N+n实例/备用服务
- 检测阶段
  - 立体化监控：机器监控/健康检查/异常监控/JVM/qps/业务监控等
  - 错误处理：业务处理与异常处理分离
  - 报警：邮件/手机分级告警等


Idempotency
BASE
Micro Service
Loose Coupling
Asynchronous Communication
Location Transparency
Event-Driven
Stateless
Shed Load
Isolation
Bulkheads
Fan out & quickest reply
Bounded Queues
Circuit Breaker
Latency Control
Fail Fast
Timeouts
Supervision
Error Handler
Full Monitor
Alarm

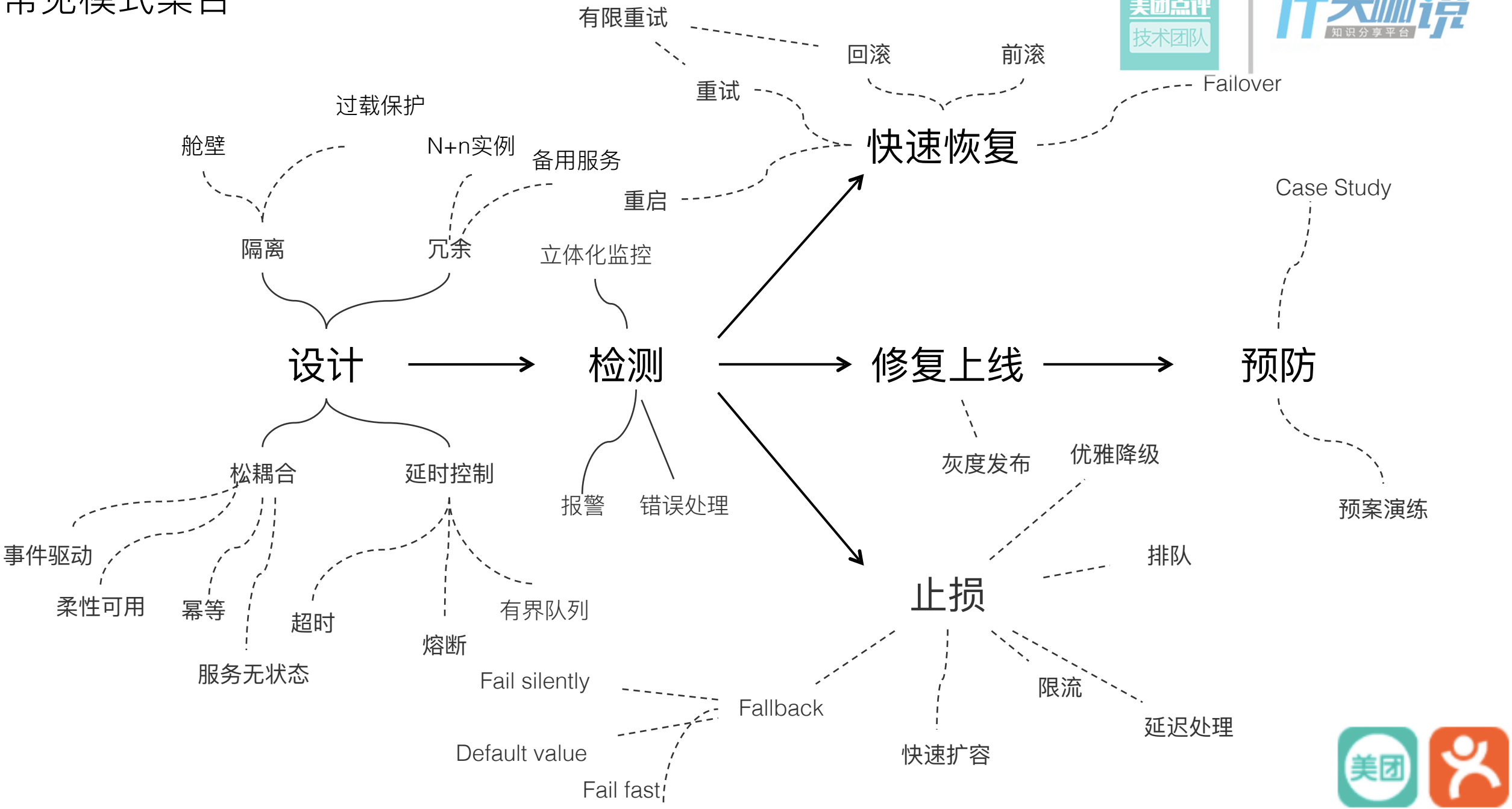
# 事中处理


有限重试
回滚
前滚
重试
Failover
快速恢复
重启
设计
检测
修复上线
预防
灰度发布
优雅降级
排队
止损
Fail silently
Fallback
限流
Default value
快速扩容
延迟处理
Fail fast

# 事后



# 常见模式集合


有限重试
回滚
前滚
重试
Failover
过载保护
舱壁
N+n实例
备用服务
快速恢复
重启
隔离
冗余
立体化监控
Case Study
设计
检测
修复上线
预防
松耦合
延时控制
灰度发布
优雅降级
报警
错误处理
预案演练
事件驱动
柔性可用
幂等
超时
熔断
有界队列
排队
止损
服务无状态
Fail silently
Fallback
限流
Default value
快速扩容
延迟处理
Fail fast

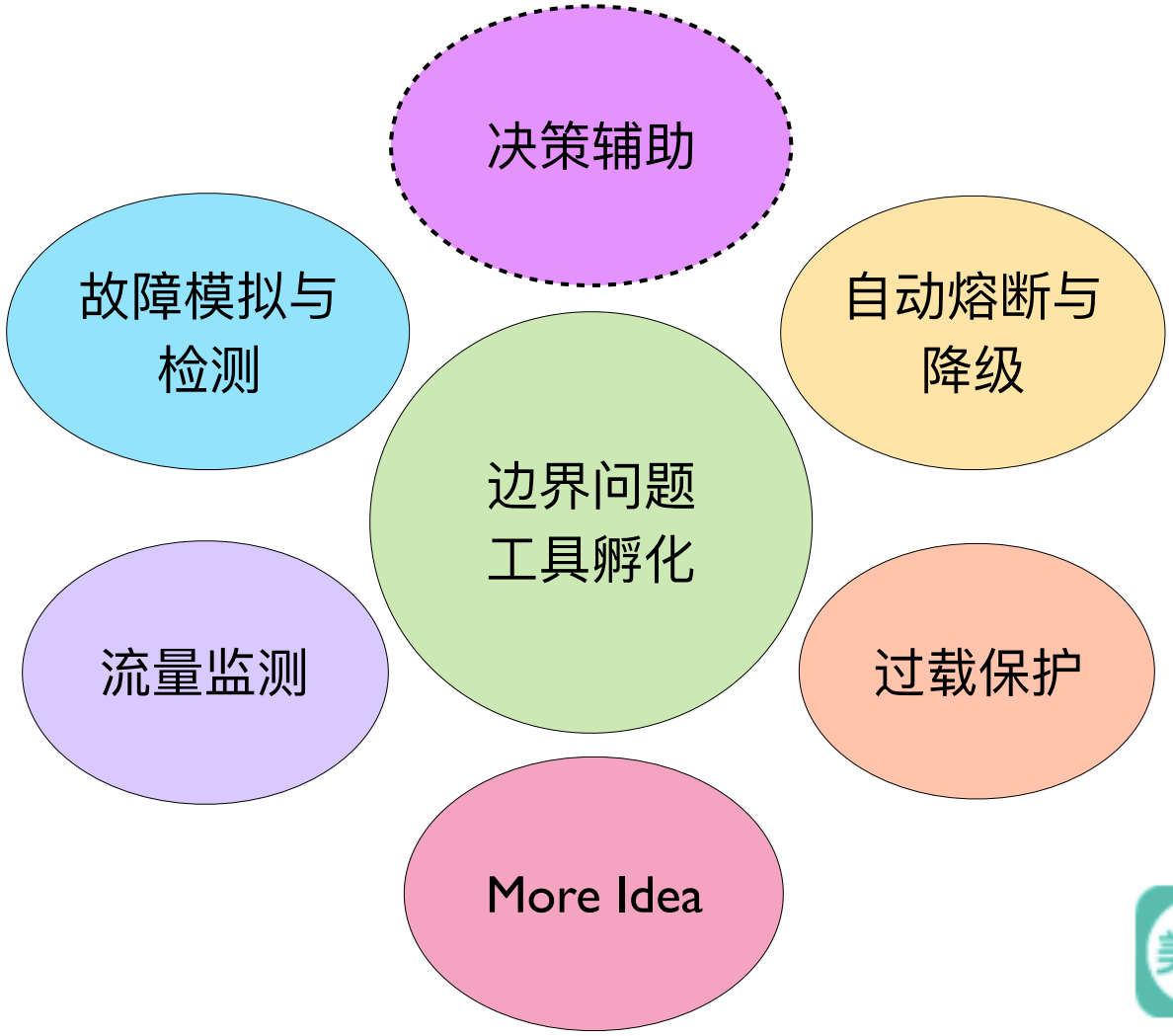
# 真实案例–构建预案

1. 架构梳理：系统里有个A服务调用了B服务，B服务属于非关键服务，通过thrift协议通信，在同一个线程中调用。
2. 提出一种边界场景：B服务出现大量超时了怎么办？该情况下会导致系统rt升高。
3. 如何监控：是否有超时异常检测？有没止损措施？Timeout模式+异常监控适合这种场景。
4. 如何补救：当B服务10秒内超时率超过30%启动自动熔断，接口 90 RT上升1.5倍时启动人工降级。Circuit breaker+ Degradation+fallback 这3种模式适合补救场景。
5. 如何预防：故障模拟演练！避免失败最好的办法就是经常失败！
6. 完善架构：检查超时时间是否合理？是否有熔断降级开关？故障模拟演练是否自动化？

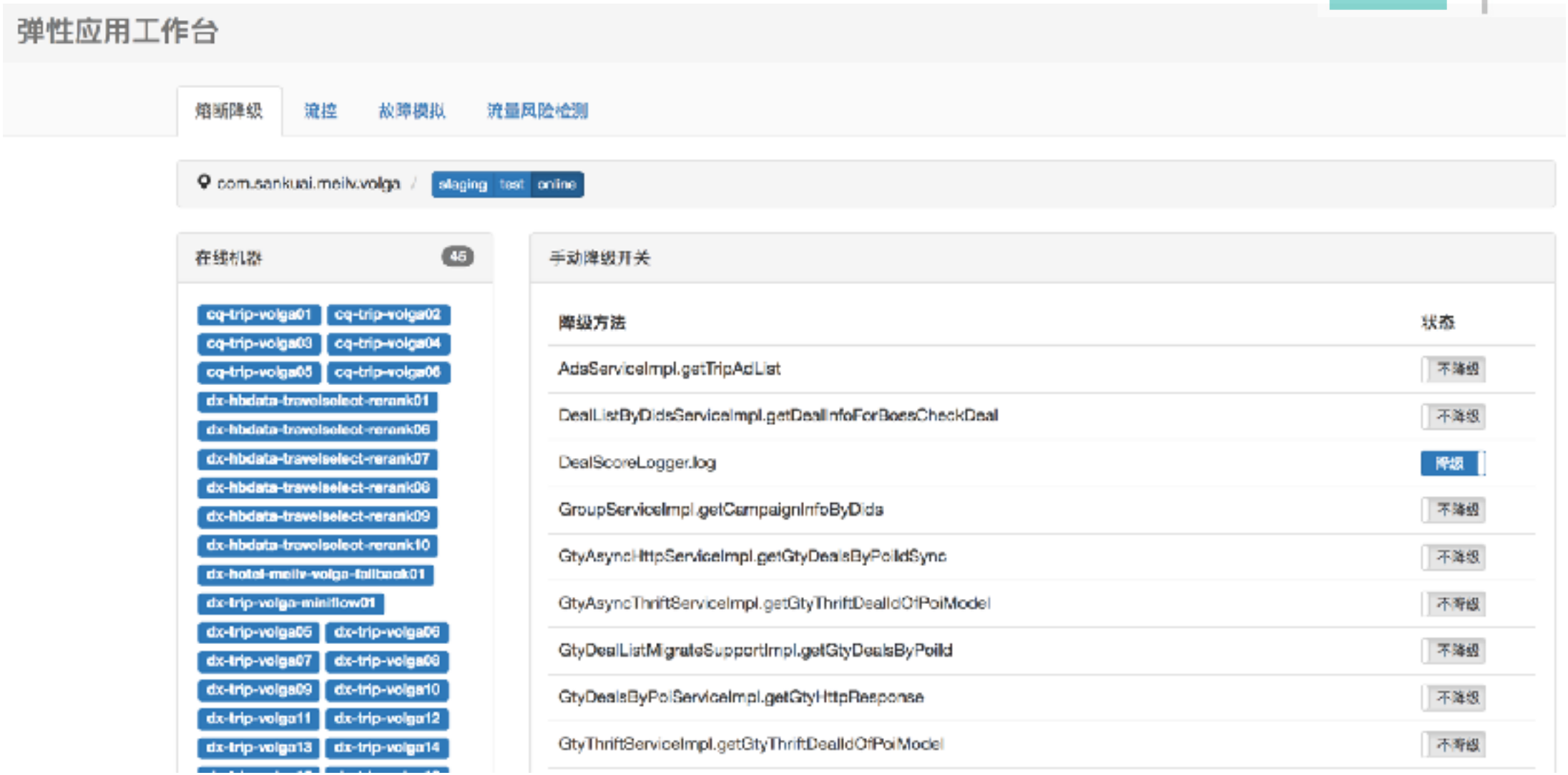
# 工具篇

# 弹性应用平台–边界问题工具套件

- 自动熔断与降级
- 过载保护
- 流量监测
- 故障模拟与预案演练
- 线程池监控



# 弹性应用平台


弹性应用工作台
熔断降级
流控
故障模拟
流量风险检测
com.sankuai.meilv.volga
staging test online
在线机器 45
cq-trip-volga01 cq-trip-volga02
cq-trip-volga03 cq-trip-volga04
cq-trip-volga05 cq-trip-volga06
dx-hbdata-travelselect-rerank01
dx-hbdata-travelselect-rerank06
dx-hbdata-travelselect-rerank07
dx-hbdata-travelselect-rerank08
dx-hbdata-travelselect-rerank09
dx-hbdata-travelselect-rerank10
dx-hotel-meilv-volga-fallback01
dx-trip-volga-miniflow01
dx-trip-volga05 dx-trip-volga06
dx-trip-volga07 dx-trip-volga08
dx-trip-volga09 dx-trip-volga10
dx-trip-volga11 dx-trip-volga12
dx-trip-volga13 dx-trip-volga14
手动降级开关
降级方法
状态
AdsServiceImpl.getTripAdList
DealListByDidsServiceImpl.getDealInfoForBoesCheckDeal
DealScoreLogger.log
GroupServiceImpl.getCampaignInfoByDids
GtyAsyncHttpServiceImpl.getGtyDealsByPoiIdSync
GtyAsyncThriftServiceImpl.getGtyThriftDealIdOfPoiModel
GtyDealListMigrateSupportImpl.getGtyDealsByPoiId
GtyDealsByPoiServiceImpl.getGtyHttpResponse
GtyThriftServiceImpl.getGtyThriftDealIdOfPoiModel
不降级
降级

# 弹性应用平台



Dashboard

应用展示 | 降级开关管理 | 流控配置 | 故障模拟 | 流量监测

Agent:jar

- 应用注册
- 熔断降级
- 流量控制
- 流量监测
- 故障注入

熔断降级数据统计

flume | kafka | storm | mafka

统计server

数据统计

流量监测数据实时处理

flume | kafka | storm | mafka

flow-detect

- 流量分析
- 流量预警

KV

redis | tair

注册配置中心

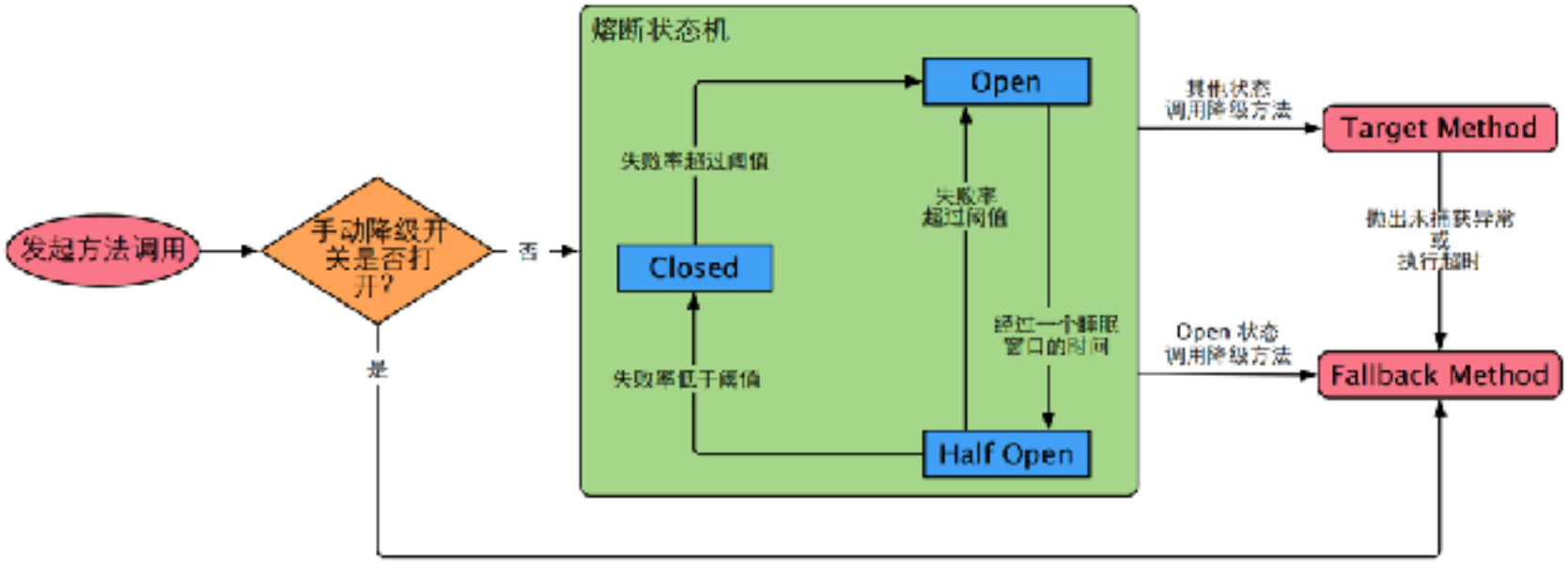
应用管理 | 配置管理

决策辅助

- 规则引擎
- 文档建设

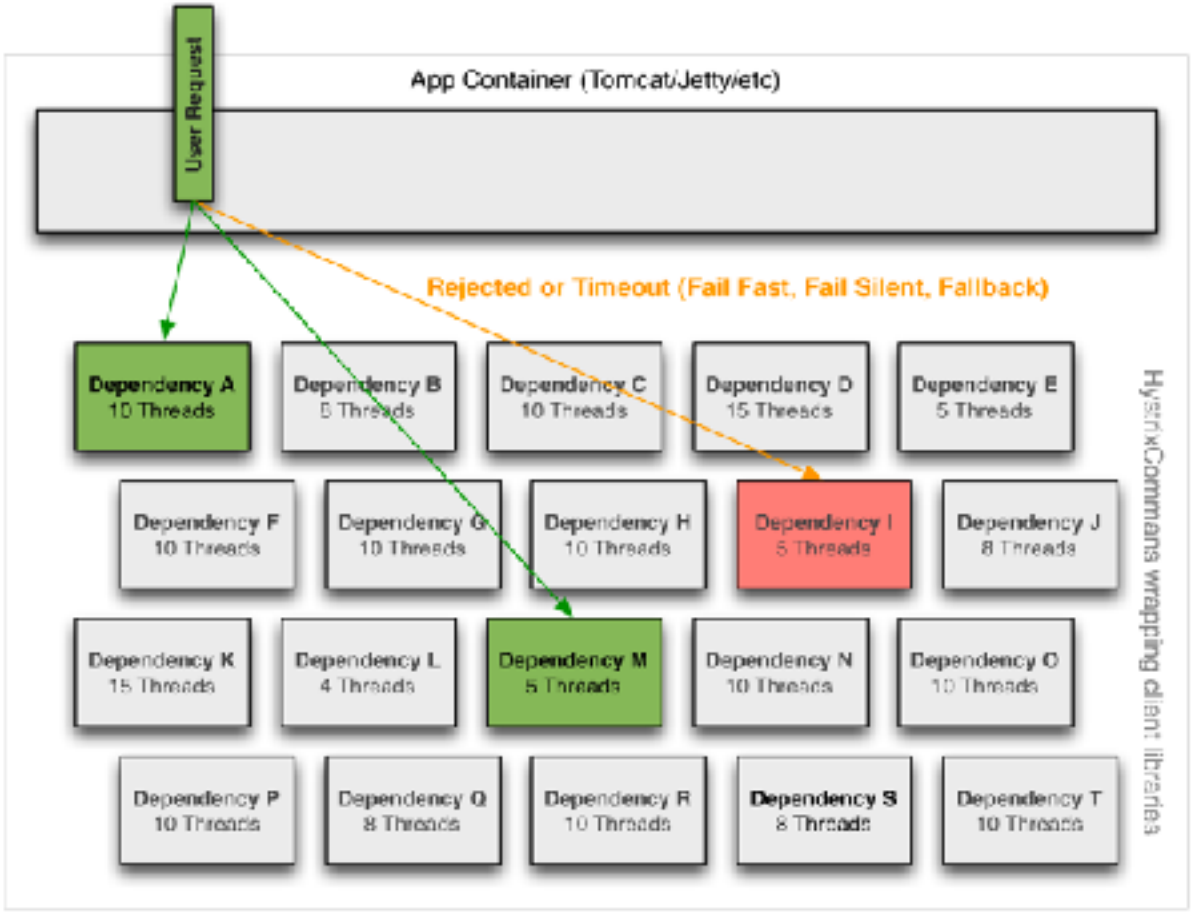
# 熔断降级组件

- 边界场景
  - 依赖服务异常/超时
  - 基础设施故障
  - 自愈
- 核心技术实现点
  - zookeeper
  - hystrix 熔断状态机
  - storm实时数据收集



# 熔断降级组件–Hystrix

- 隔离模式
  - 线程池
  - 信号量
  - NONE



# 熔断降级组件–Hystrix

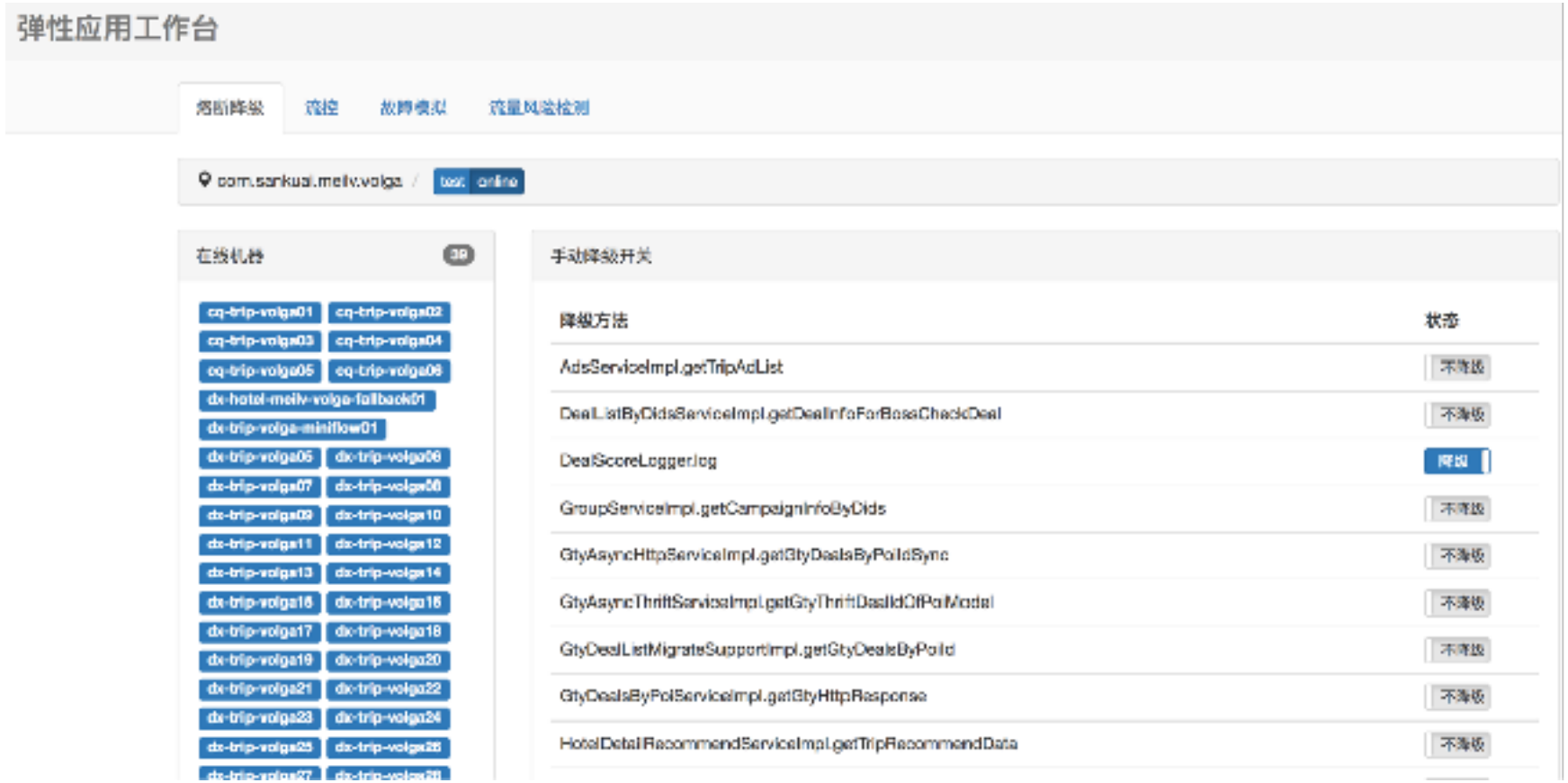
```
@Override
@HystrixCommand(groupKey = "userType", commandKey = "getUserTicketTypeInfo", fallbackMethod ="getUserTicketTypeInfoFallback",
commandProperties = {
@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "2000"),
@HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value ="10"),
@HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "50"),
@HystrixProperty(name = "circuitBreaker.manualControl", value = "true")

})
public UserTypeRelation getUserTicketTypeInfo(String userId) {
    return getUserType(userId, url +"usertype/v1/get");
}

public UserTypeRelation getUserTicketTypeInfoFallback(String userId, Throwable ex){
    LOGGER.error("get usertype fallback: {}", userId, ex);
    return UserTypeRelation.NEW;
}
```

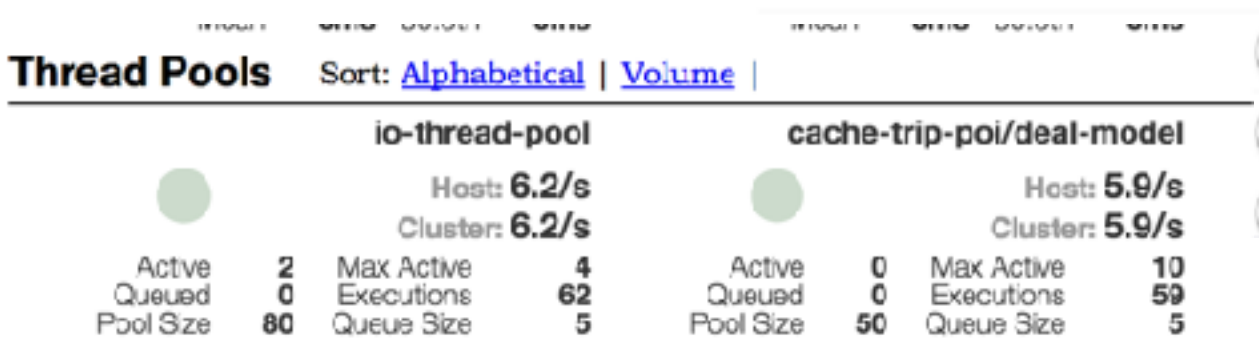
# 熔断降级组件–Hystrix对于性能的影响

| 并发: 100, 各隔离策略方法循环 10000 次采集耗时数据 | | | | | |
|---|---|---|---|---|---|
| 隔离策略 | 最低耗时(μs) | TP50(μs) | TP90(μs) | TP999(μs) | 平均耗时(μs) |
| 未接入 | 0 | 0 | 0 | 9 | 0.1123 |
| SEMAPHORE | 33 | 77 | 356 | 1731 | 157.8358 |
| NONE | 33 | 71 | 189 | 1553 | 114.3972 |
| THREAD | 52 | 117 | 415 | 2773 | 222.468 |


弹性应用工作台
熔断降级
流控
故障模拟
流量风险检测
com.sankuai.meituan.volga / test online
在线机器 39
手动降级开关
降级方法
状态
AdsServiceImpl.getTripAdList
DealListByDidsServiceImpl.getDealInfoForBosaCheckDeal
DealScoreLogger.log
GroupServiceImpl.getCampaignInfoByDids
GtyAsyncHttpServiceImpl.getGtyDealsByPoiIdSync
GtyAsyncThriftServiceImpl.getGtyThriftDealIdOfPoiModel
GtyDealListMigrateSupportImpl.getGtyDealsByPoiId
GtyDealsByPoiServiceImpl.getGtyHttpResponse
HotelDetailRecommendServiceImpl.getTripRecommendData

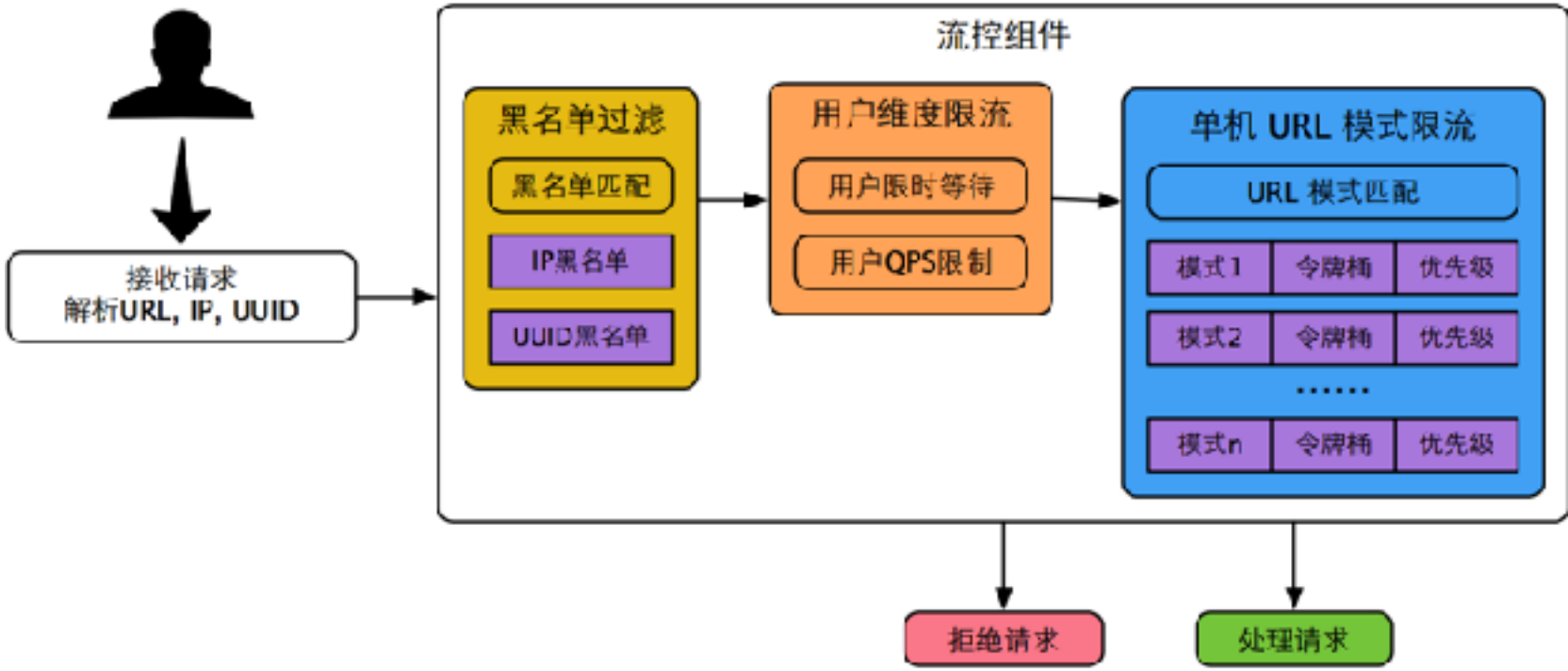

美团点评
技术团队
返回
IT大咖说
知识分享平台
online
在线机器 39
手动降级开关 38
AdsServiceImpl.getTripAdList
DealListByDidsServiceImpl.getDe...
DealScoreLogger.log
GroupServiceImpl.getCampaignI...
GtyAsyncHttpServiceImpl.getGty...
GtyAsyncThriftServiceImpl.getGt...
GtyDealListMigrateSupportImpl.g...


Circuit
Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Media
getUserTicketTypeInfo
0.0 %
Host: 11.8/s
Cluster: 11.8/s
Circuit Closed
Median 1ms 99th 9ms
Mean 1ms 99.5th 9ms
hbDataRecommendClient
0.0 %
Host: 9.4/s
Cluster: 9.4/s
Circuit Closed
Median 13ms 99th 39ms
Mean 13ms 99.5th 38ms
getThriftDealIdOfPoiModel
0.0 %
Host: 2.6/s
Cluster: 2.6/s
Circuit Closed
getTcDealsByPoiIdSync
0.0 %
Host: 2.8/s
Cluster: 2.8/s
Circuit Closed

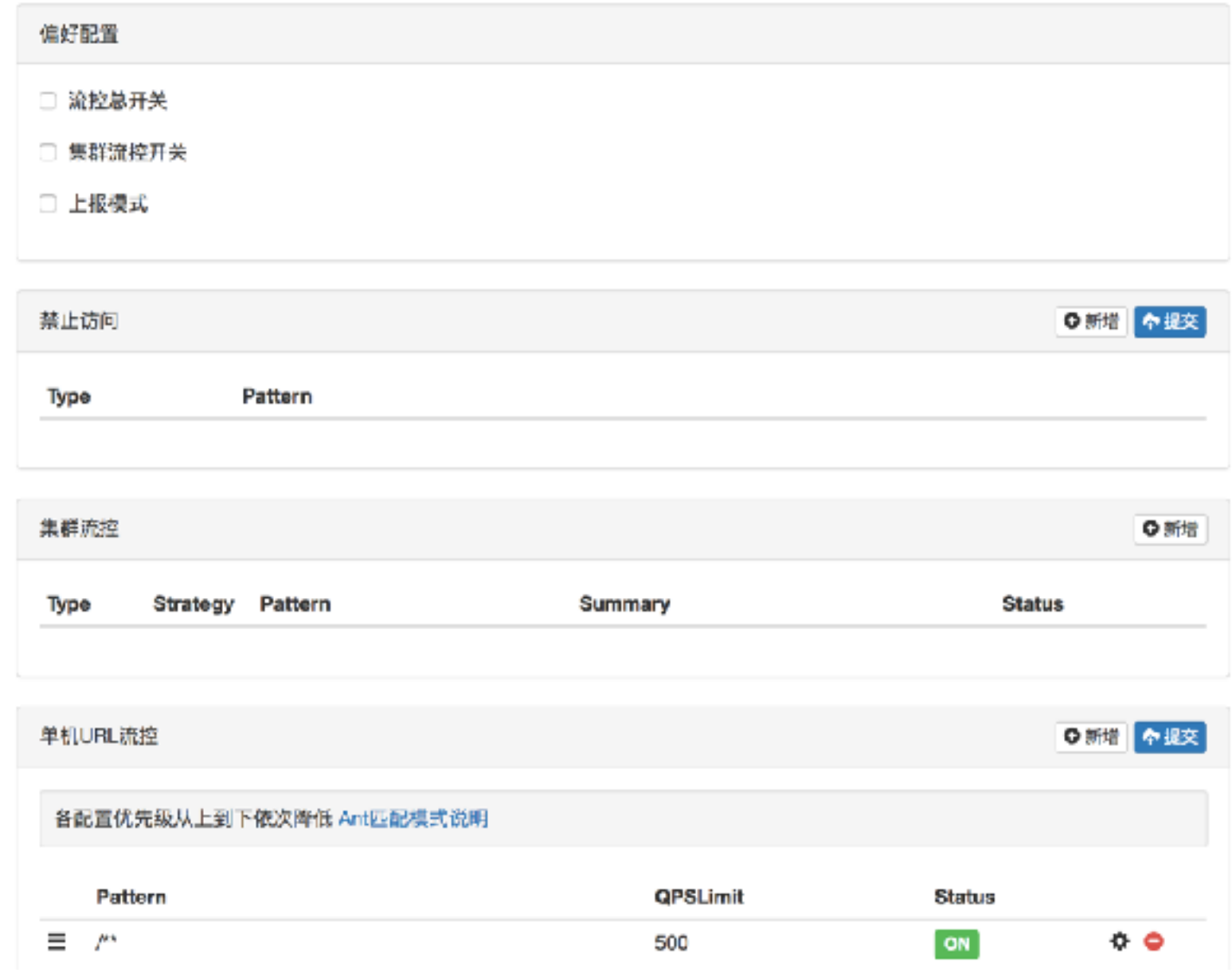

Thread Pools
Sort: Alphabetical | Volume |
io-thread-pool
Host: 6.2/s
Cluster: 6.2/s
Active 2
Queued 0
Pool Size 80
Max Active 4
Executions 62
Queue Size 5
cache-trip-poi/deal-model
Host: 5.9/s
Cluster: 5.9/s
Active 0
Queued 0
Pool Size 50
Max Active 10
Executions 59
Queue Size 5

# 过载保护–流控

- 边界场景
  - 恶意爬虫
  - 流量过载
  - 频率控制
- 核心技术实现点
  - zookeeper
  - Redis
  - guava/ant matcher

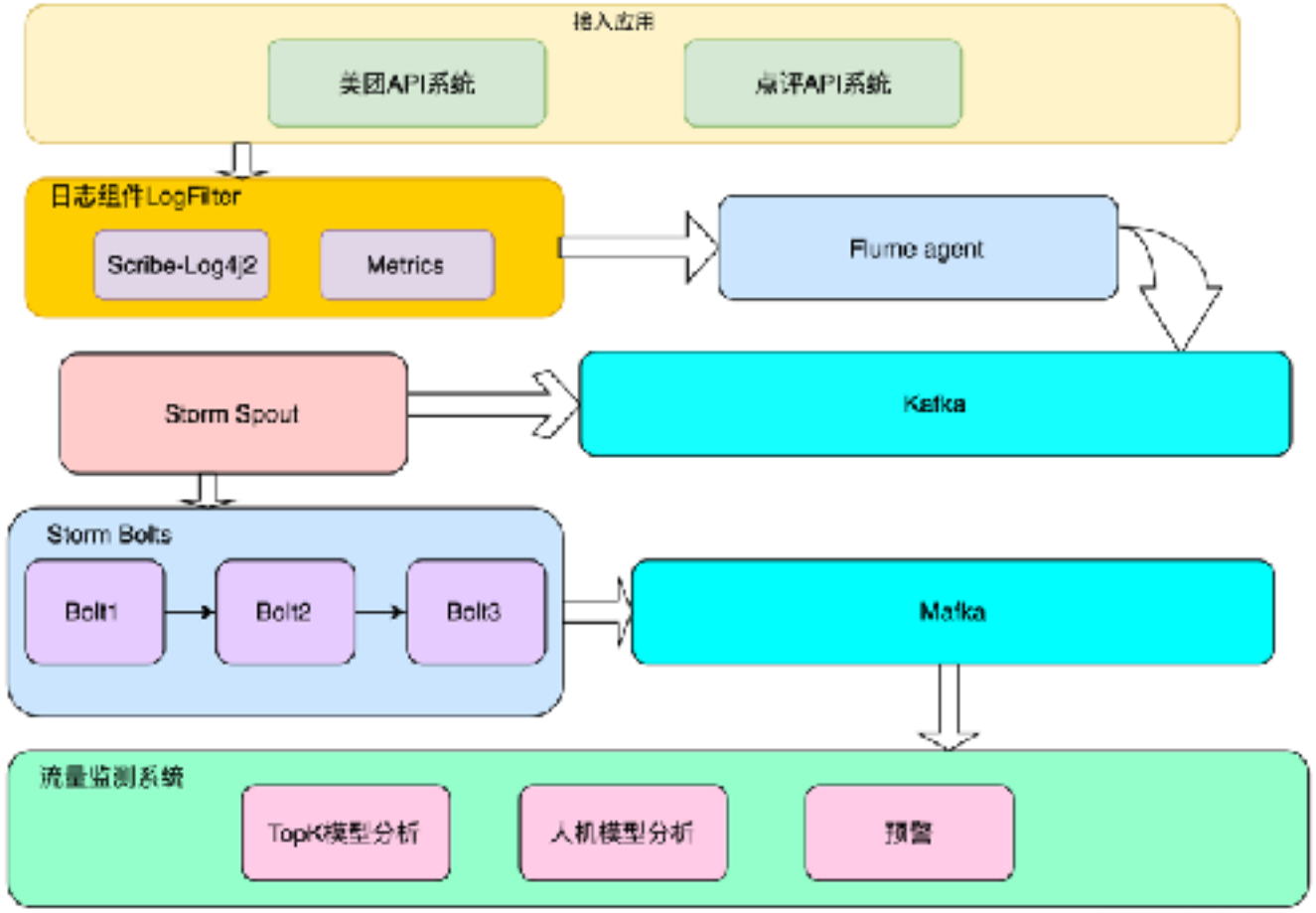


# 过载保护–流控


偏好配置
流控总开关
集群流控开关
上报模式
禁止访问
新增
提交
Type
Pattern
集群流控
新增
Type
Strategy
Pattern
Summary
Status
单机URL流控
新增
提交
各配置优先级从上到下依次降低 Ant匹配模式说明
Pattern
QPSLimit
Status
/**
500
ON

# 流量监测与分析

- 边界场景
  - 异常流量，如爬虫
  - 人机识别
- 核心技术实现点
  - 基于storm的实时日志分析
  - 算法模型

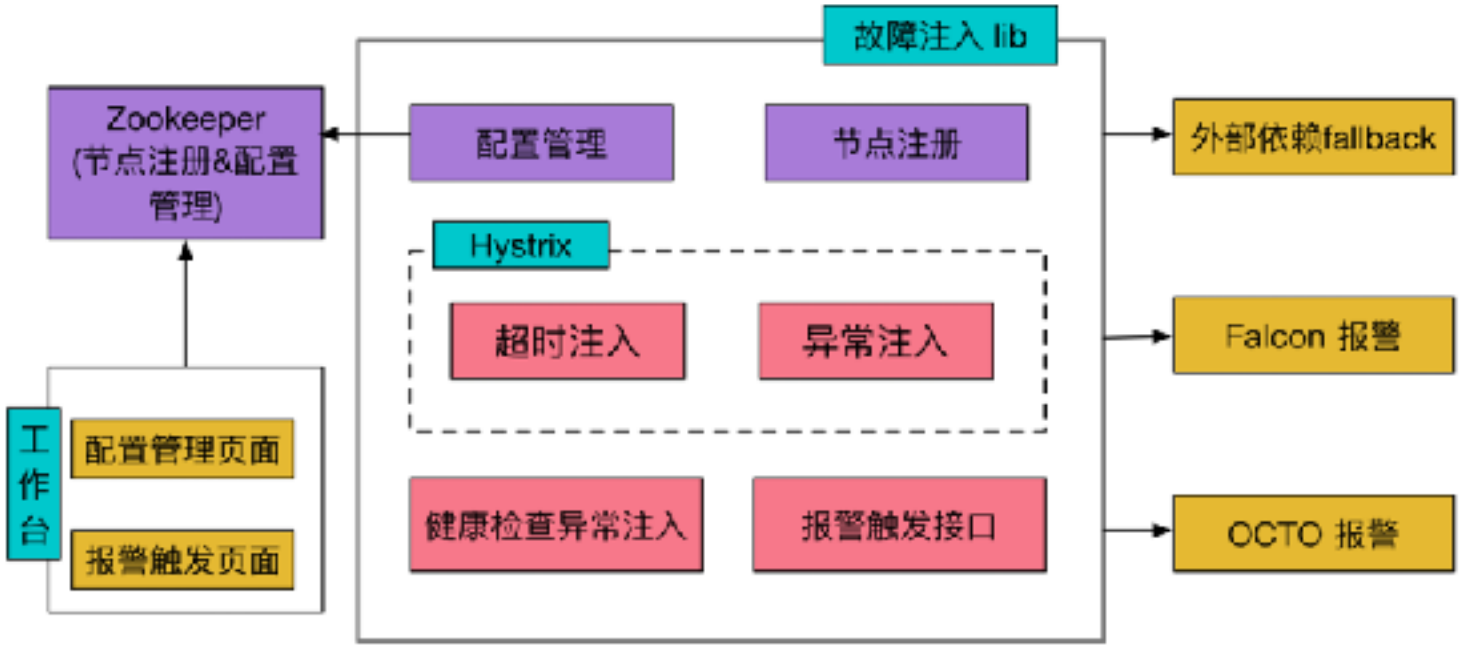


# 故障模拟与注入–猴子家族

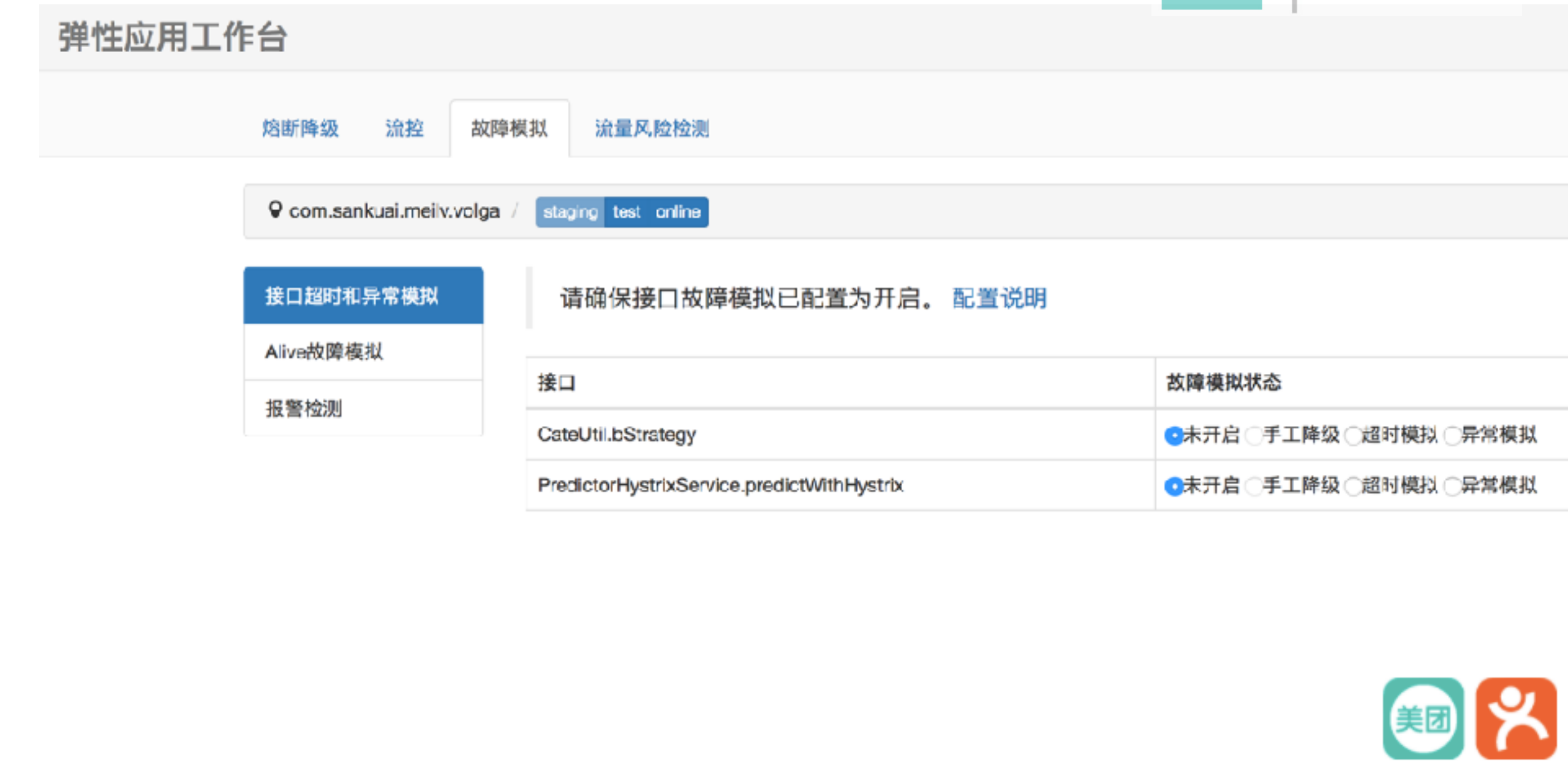
- SimianArmy of Netflix
  - 硬件故障
  - 网络故障
  - 服务故障

# 故障注入与模拟

- 场景
  - 预案演练：服务超时/异常
  - 定期模拟检测
- 核心技术
  - hystrix
  - rxjava
  - zookeeper



# 故障注入与模拟


弹性应用工作台
熔断降级
流控
故障模拟
流量风险检测
com.sankuai.meituan.volga
staging
test
online
接口超时和异常模拟
Alive故障模拟
报警检测
请确保接口故障模拟已配置为开启。
配置说明
接口
故障模拟状态
CateUtil.bStrategy
未开启
手工降级
超时模拟
异常模拟
PredictorHystrixService.predictWithHystrix
未开启
手工降级
超时模拟
异常模拟

# 最后的建议

# 适用场景

- 追求99.99%的可用性的系统
- 拥有大规模的分布式服务的团队

# 未来的演进

- 行业的趋势：Netflix/Linkedin/ElasticSearch/Facebook　在部分开源代码中开始将弹性（Resilience）列为系统的重要能力，成为业内的可用性领域的一个关注趋势！
- 未来规划：集成更多的边界场景的解决方案。

# 参考资料

- https://github.com/Netflix/Hystrix/wiki
- 《微服务设计》
- 《Site Reliability Engineering》
- 《反脆弱:从不确定性中获益》

Do not avoid failures. Embrace them!

杀不死我的，只会让我更坚强！

美团点评 技术团队

IT大咖说 知识分享平台