

链路稳定性探测技术方案

文档历史

修订日期	修订内容	修订版本	修订人
2022-02-14	创建文档	v0.1	lin.zhu@shopee.com
2022-02-17	增加系统目标分析	v0.2	lin.zhu@shopee.com
2022-02-21	更新目标分析和架构分析	v0.3.1	lin.zhu@shopee.com
2022-02-22	更新核心业务规则和实现	v0.3.2	lin.zhu@shopee.com
2022-02-27	更新系统设计目标和架构图	v0.3.3	lin.zhu@shopee.com
2022-03-03	更新系统设计目标和架构图	v0.3.4	lin.zhu@shopee.com
2022-03-17	更新核心业务规则中的代码分析实现及分析报告的模板	v0.3.5	liangming.huang@shopee.com
2022-03-18	更新核心业务规则中的DB分析实现	v0.3.6	liangming.huang@shopee.com
2022-03-18	更新混沌工程设计方案	v0.3.7	yiran.jin@shopee.com

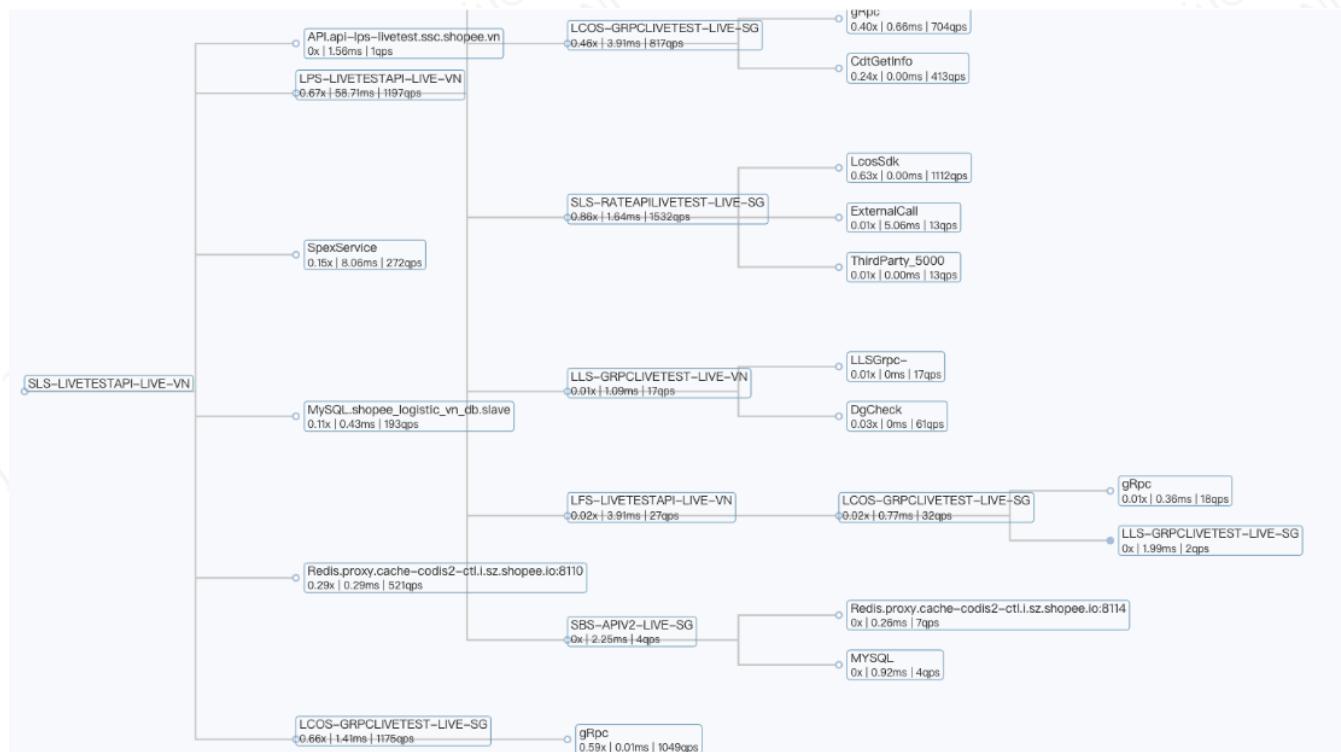
摘要

编写目的

对链路稳定性探测-混沌工程实现的分析。为链路稳定性探测的开发和测试提供技术参考。

项目背景

随着服务拆分，系统链路变长，系统可能出错的地方也随之增加。一个常见的系统链路如下：



本项目整理集成了常见问题检查，为系统稳定性评价提供判断依据。

参考资料

[混沌工程\(Chaos Engineering\) 总结](#)

[阿里巴巴混沌测试工具ChaosBlade](#)

[字节跳动混沌工程实践总结](#)

[混沌工程原则](#)

[Golang 常见性能问题总结](#)

任务概述

对于链路稳定性探测，主要需要实现以下功能：

任务	描述
链路检测	检测和展示系统链路
链路稳定性分析	分析链路中隐患和强弱依赖

规范与约定

术语与缩略语

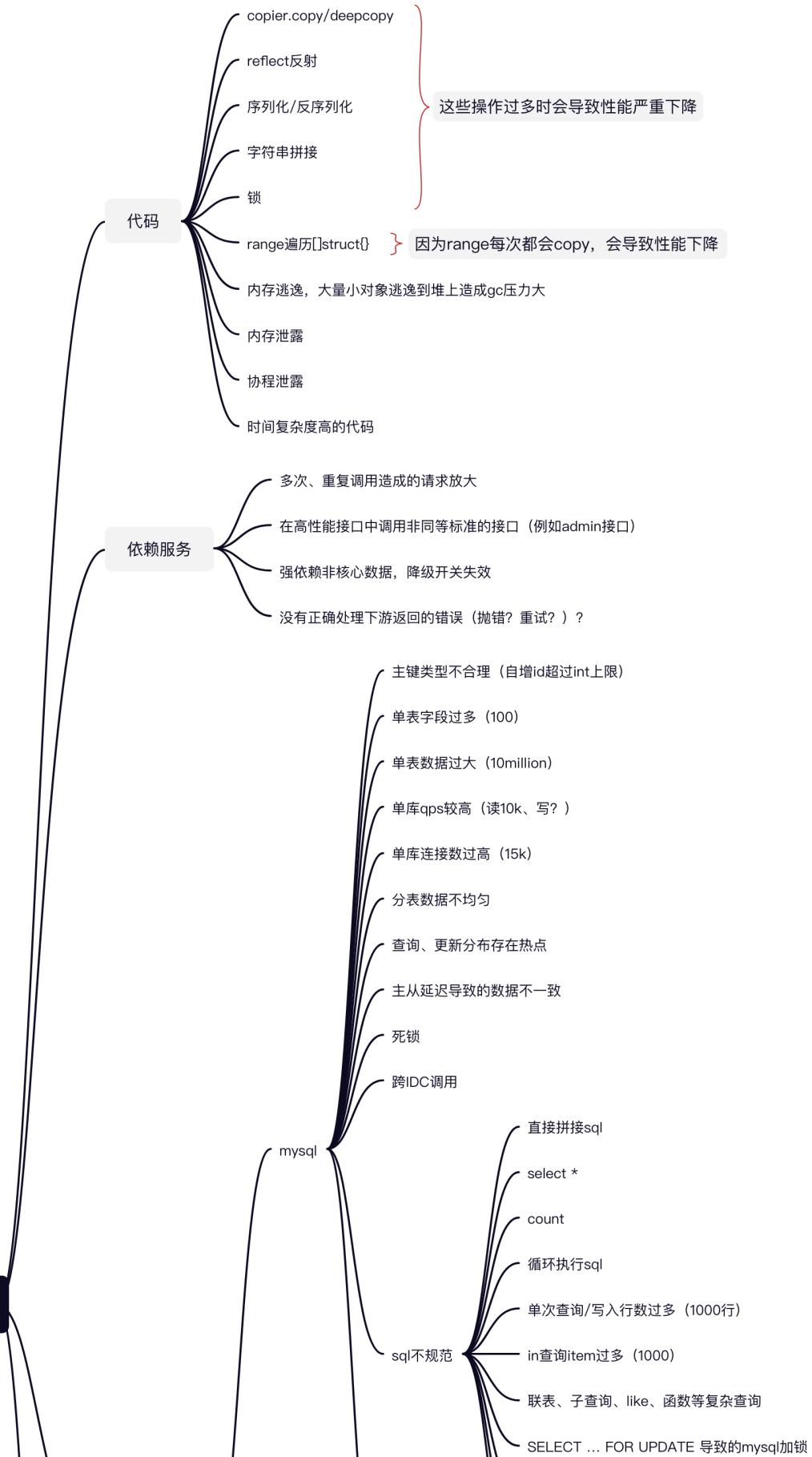
缩略语/术语	全称	说明
链路		系统中业务逻辑从开始到结束所需要经过的子系统组成的调用链路
混沌工程		由netflix工程师提出的，旨在随机注入故障来验证系统稳定性的工具
handler		chassis中的handler，是可以配置在服务入口、下游客户端的拦截器

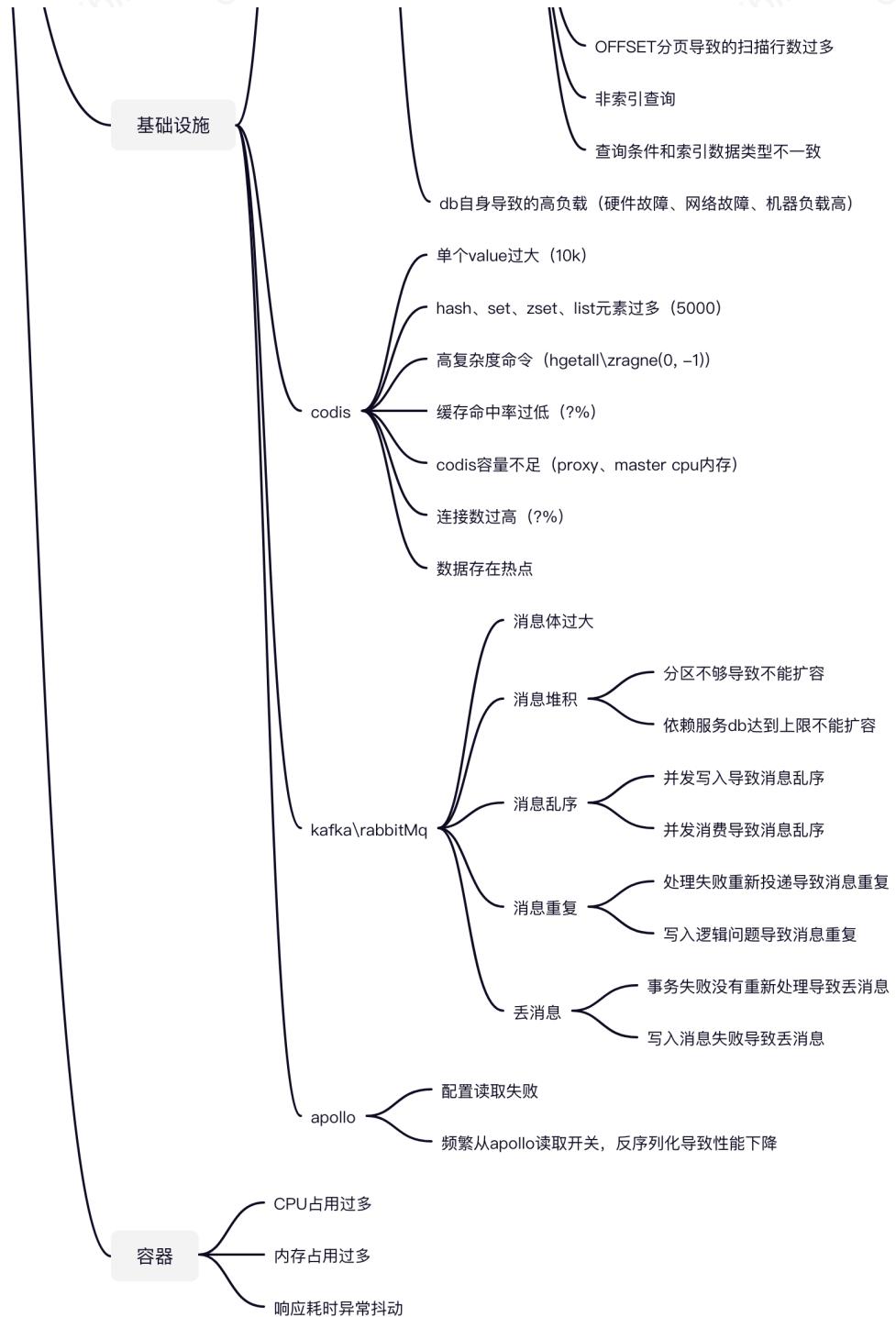
系统分析设计

系统设计目标

首先分析稳定性的影响因素，分为代码、依赖服务、基础设施、容灾能力四部分：

常见稳定性问题





系统的设计目标是从上述影响因素入手，自动检测分析系统的稳定性。

总体架构分析

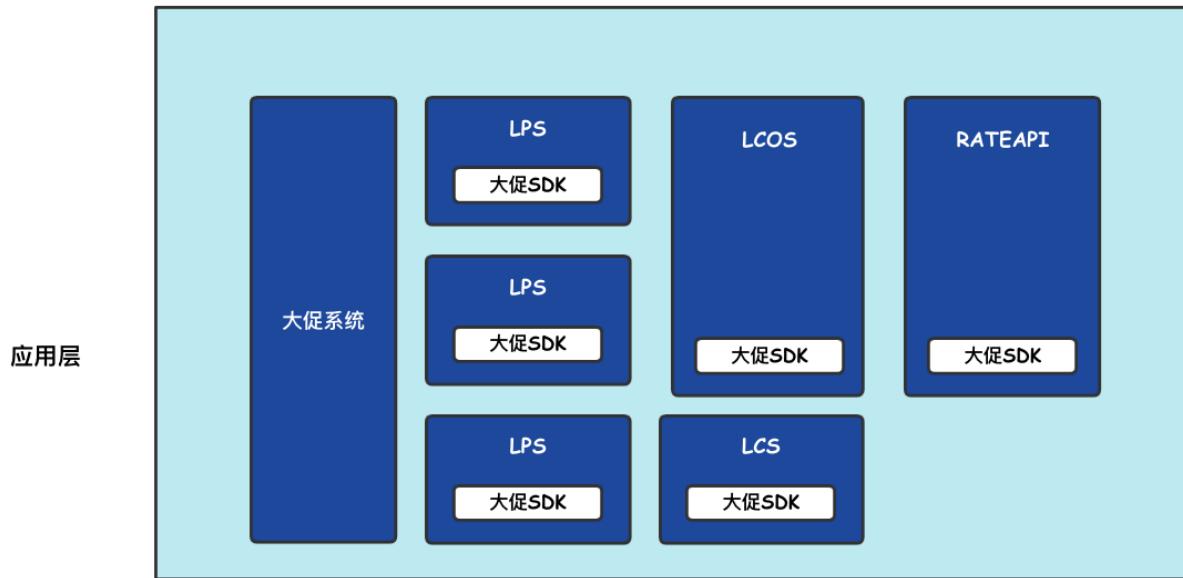
稳定性分析是大促系统的一部分，大促系统架构如下：

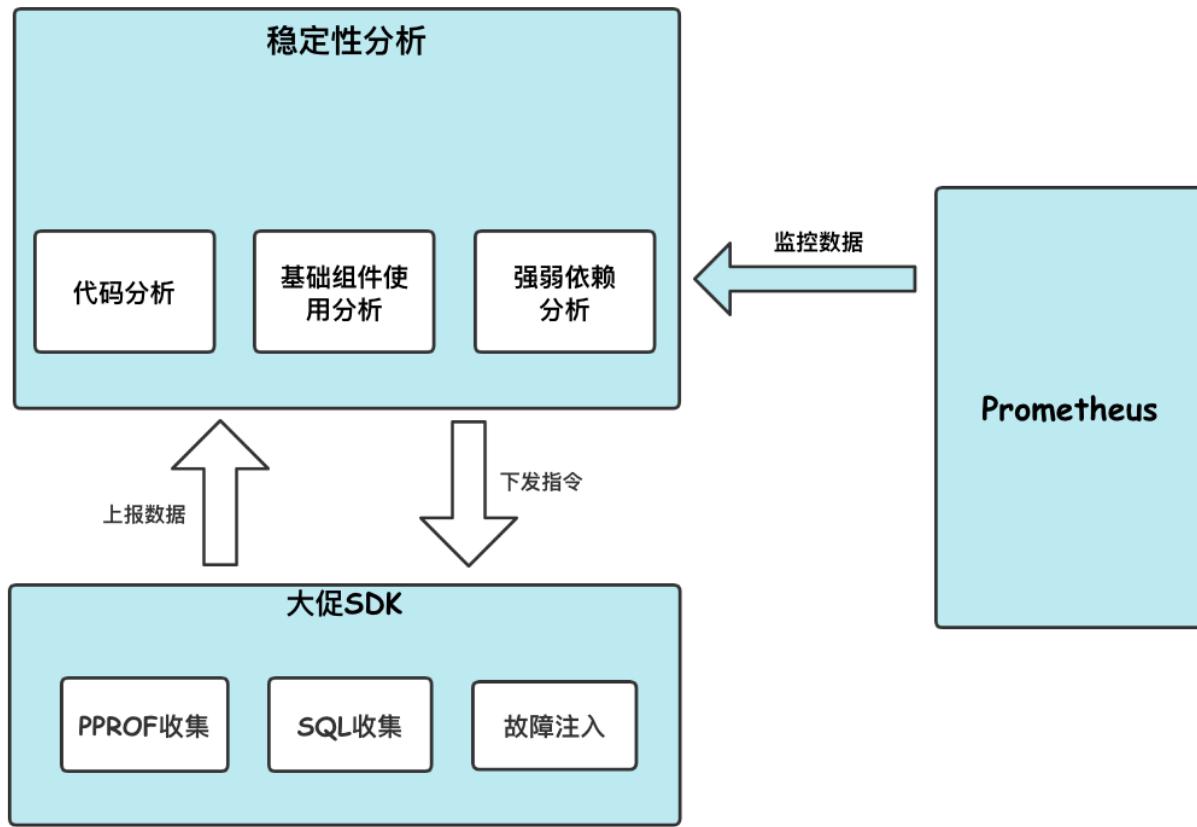


根据设计目标，本项目主要包含以下模块，

- 强弱依赖分析
- 代码分析
- 基础组件使用分析

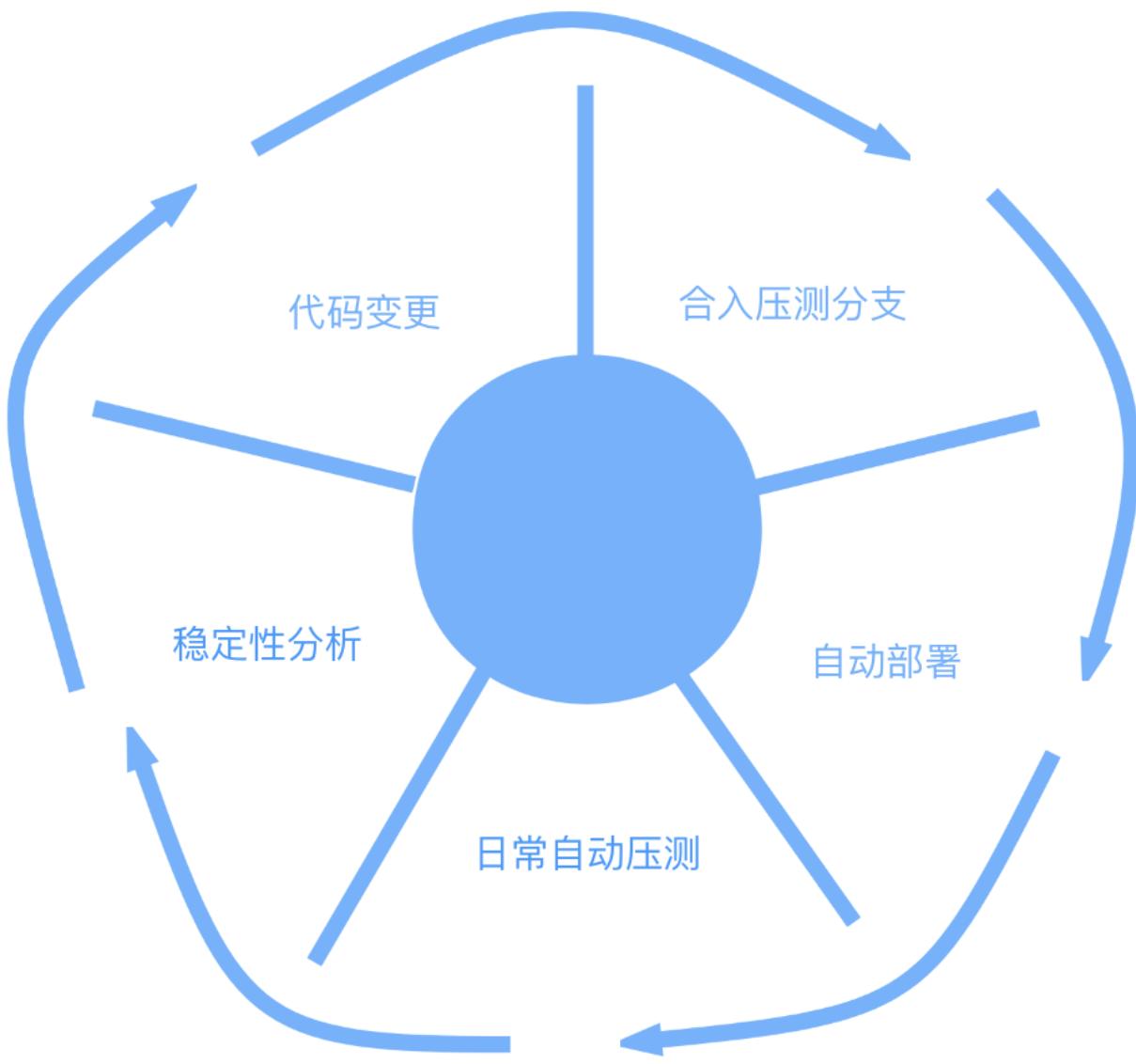
以上分析模块都需要采集系统数据或到系统中执行操作，这里使用服务内嵌像cat-client这样的大促SDK的方式实现(如有必要的话)：



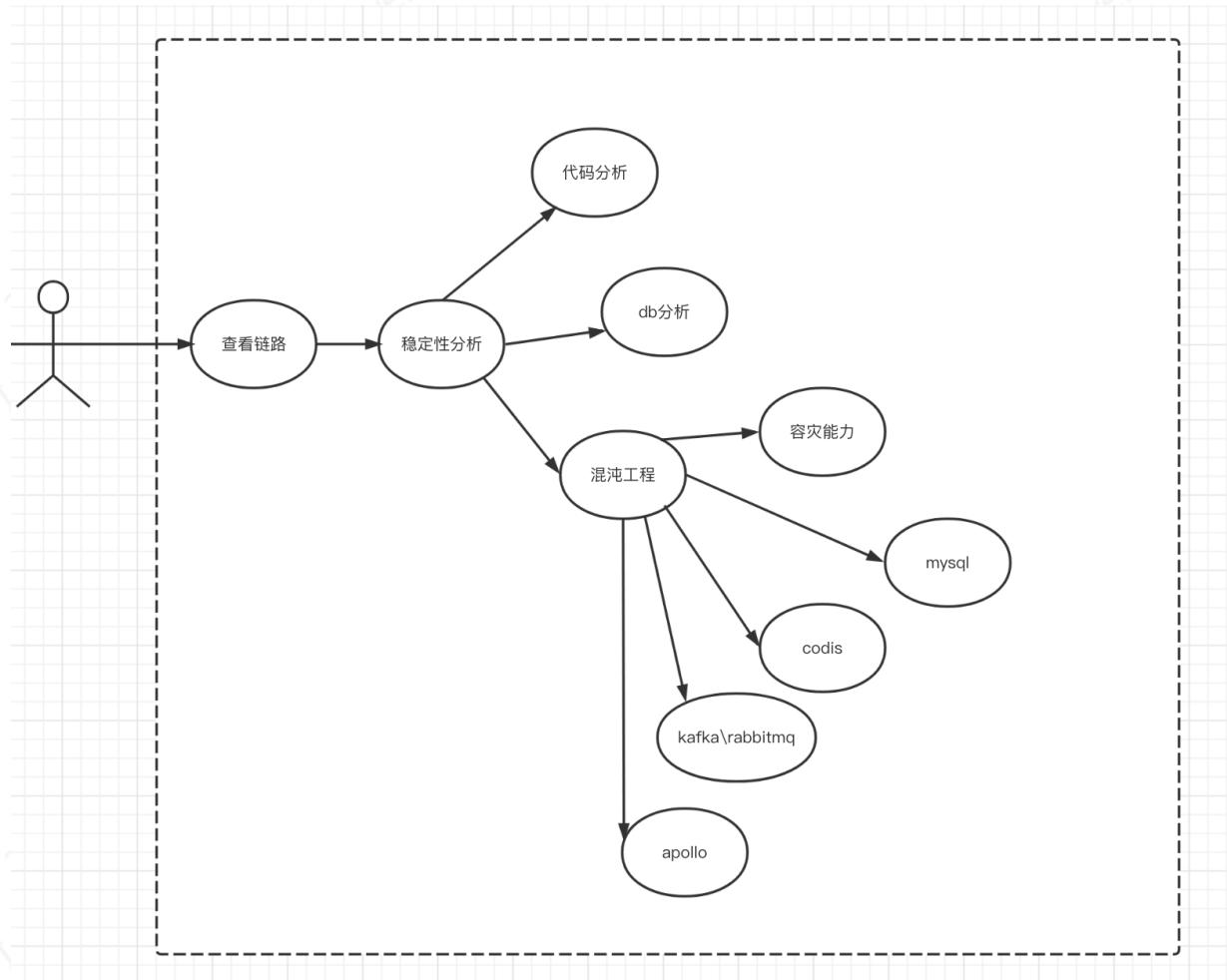


SDK收集服务运行数据上报到大促系统进行分析，同时SDK接收指令触发实验故障以实现混沌工程分析。

整个分析过程和Livetest环境日常自动压测相结合，在压测分支接入大促SDK，使得整个分析过程不影响线上服务。



用例分析



如上图所示，在链路中启动稳定性分析后，系统依次执行代码分析、db分析、混沌工程，并生成分析报告。

核心业务规则

下面具体描述稳定性分析各项用例的实现。

代码分析

sdk采集pprof上报，大促系统通过分析pprof检测是否存在常见性能瓶颈。

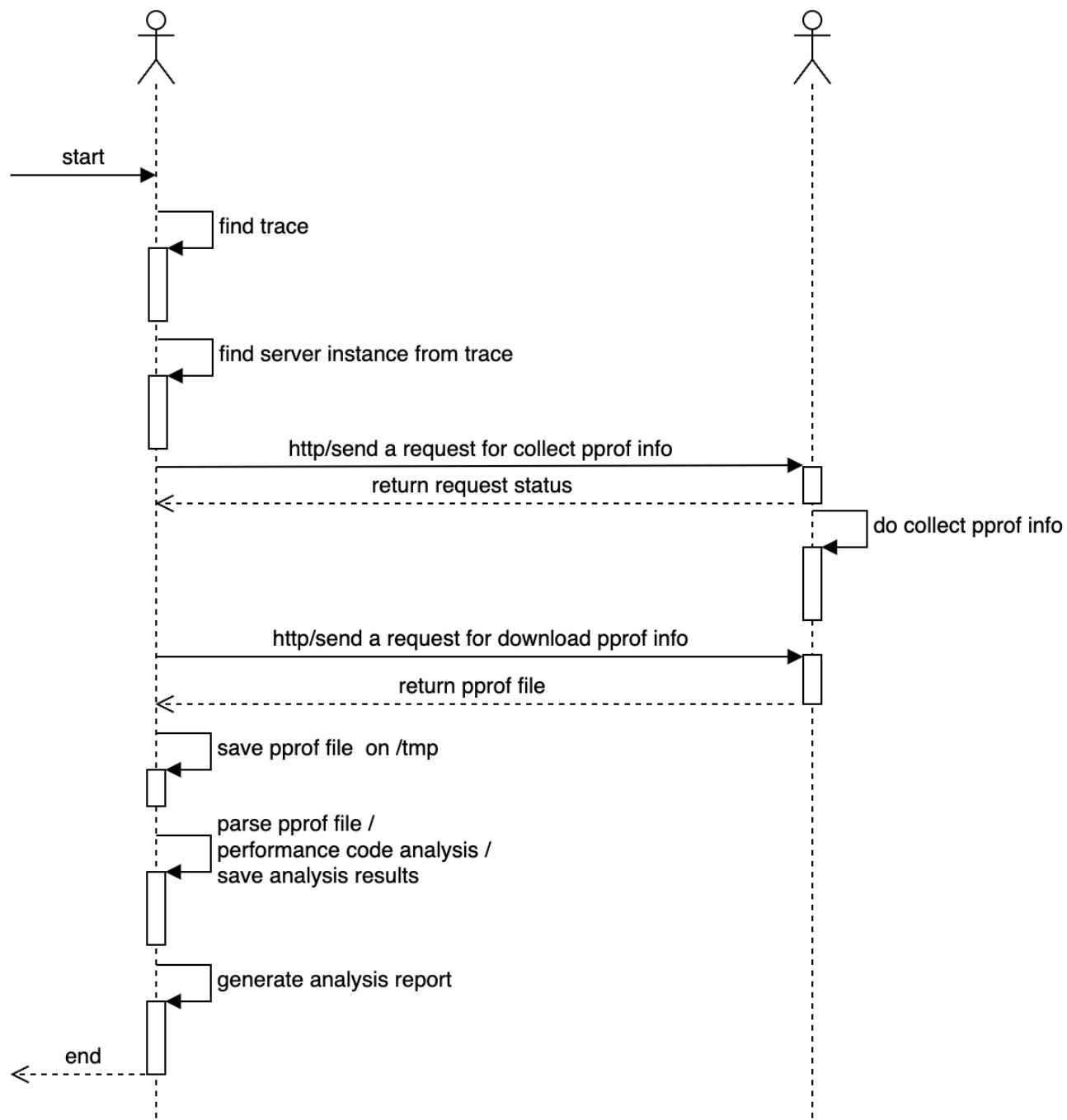
代码分析主要有三部分：pprof信息收集、pprof信息分析和生成分析报告。

pprof信息采集方案：打通k8s，使用k8s的pprof生成接口和下载接口获取

代码分析时序图：

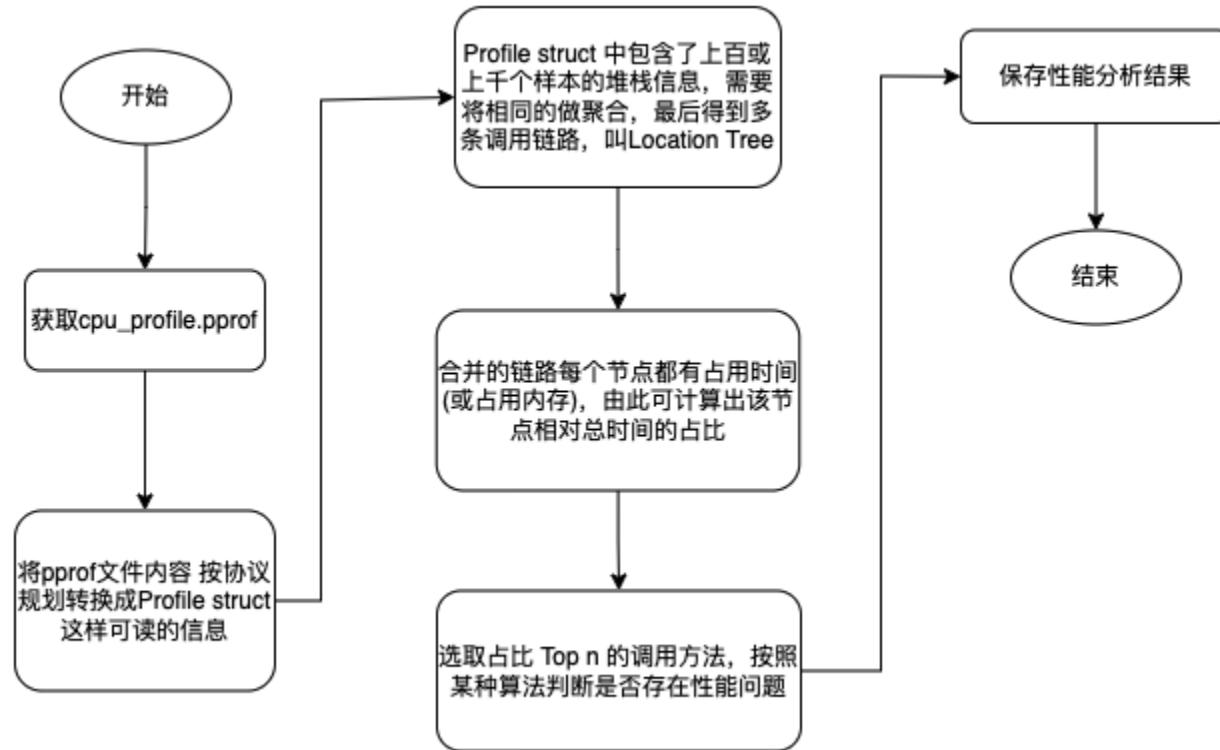
Celestial Server

K8S



该方案实现了对目标系统的最少侵入，利用k8s现有的功能就可以获取得到服务的pprof文件，以用于分析，但前提是k8s能够提供pprof 的生成和下载的对外接口，当前不支持。

pprof信息分析与 go tool pprof 工具类似，通过解析pprof 文件得到 profile 对象，聚合所有样例对象，计算每个location 的时间或内存的占用情况。



go tool pprof 的分析结果：

pprof						VIEW	SAMPLE	REFINE	CONFIG	Search regexp
Flat	Flat%	Sum%	Cum	Cum%	Name					
0	0.00%	0.00%	140ms	56.00%	runtime.systemstack					
0	0.00%	0.00%	100ms	40.00%	runtime.mstart					
0	0.00%	0.00%	70ms	28.00%	runtime.schedule					
0	0.00%	0.00%	60ms	24.00%	runtime.semalsleep					
0	0.00%	0.00%	60ms	24.00%	runtime.mcall					
0	0.00%	0.00%	60ms	24.00%	runtime.gcDrain					
0	0.00%	0.00%	60ms	24.00%	runtime.gcBgMarkWorker.func2					
0	0.00%	0.00%	60ms	24.00%	runtime.gcBgMarkWorker					
50ms	20.00%	20.00%	50ms	20.00%	runtime.kevent					
0	0.00%	20.00%	50ms	20.00%	runtime.startTheWorldWithSema					
0	0.00%	20.00%	50ms	20.00%	runtime.park_m					
0	0.00%	20.00%	50ms	20.00%	runtime.netpoll					
0	0.00%	20.00%	50ms	20.00%	runtime.gcStart.func2					
0	0.00%	20.00%	50ms	20.00%	runtime.findRunnable					
40ms	16.00%	36.00%	40ms	16.00%	runtime.pthread_cond_wait					
0	0.00%	36.00%	40ms	16.00%	runtime.stopm					
0	0.00%	36.00%	40ms	16.00%	runtime.notesleep					
0	0.00%	36.00%	30ms	12.00%	runtime.(*gcWork).balance					
20ms	8.00%	44.00%	20ms	8.00%	runtime.usleep					
20ms	8.00%	52.00%	20ms	8.00%	runtime.scanobject					

大促系统的分析结果：

```

{
  "JSON": {
    "TotalTimeSecond": 0.25,
    "TotalTime": 250000000
  },
  "Metrics": [
    {
      "0": {
        "Flat": 0,
        "FlatRatio": 0,
        "Cum": 0.14,
        "CumRatio": 56,
        "Name": "runtime.systemstack",
        "File": "/usr/local/opt/go@1.14/go1.14.15/src/runtime/asm_amd64.s"
      },
      "1": {
        "Flat": 0,
        "FlatRatio": 0,
        "Cum": 0.1,
        "CumRatio": 40,
        "Name": "runtime.mstart",
        "File": "/usr/local/opt/go@1.14/go1.14.15/src/runtime/proc.go"
      },
      "2": {
        "Flat": 0,
        "FlatRatio": 0,
        "Cum": 0.07,
        "CumRatio": 28,
        "Name": "runtime.schedule",
        "File": "/usr/local/opt/go@1.14/go1.14.15/src/runtime/proc.go"
      },
      "3": {
        "Flat": 0,
        "FlatRatio": 0,
        "Cum": 0.06,
        "CumRatio": 24,
        "Name": "runtime.semalsleep",
        "File": "/usr/local/opt/go@1.14/go1.14.15/src/runtime/os_darwin.go"
      },
      "4": {
        "Flat": 0,
        "FlatRatio": 0,
        "Cum": 0.06,
        "CumRatio": 24,
        "Name": "runtime.gcBgMarkWorker",
        "File": "/usr/local/opt/go@1.14/go1.14.15/src/runtime/mgc.go"
      }
    ]
  ]
}

```

定位到消耗性能的Location（其实是方法）后，还可以根据解析得到的文件名、方法名及行数，得到具体消耗性能的代码行。

```

{
  "JSON": {
    "Frame": {
      "Key": "google.golang.org/grpc/internal/transport.(*http2Client).reader:/root/go/pkg/mod/google.golang.org/grpc@v1.34.0/internal/transport/http2_client.go:0",
      "Name": "google.golang.org/grpc/internal/transport.(*http2Client).reader",
      "SystemName": "google.golang.org/grpc/internal/transport.(*http2Client).reader",
      "File": "/root/go/pkg/mod/google.golang.org/grpc@v1.34.0/internal/transport/http2_client.go",
      "Line": 0,
      "Col": 0,
      "SelfWeight": 60000000,
      "TotalWeight": 2360000000
    },
    "Children": [
      {
        "0": {
          "1": {
            "Frame": {
              "Key": "google.golang.org/grpc/internal/transport.(*http2Client).handlePing:/root/go/pkg/mod/google.golang.org/grpc@v1.34.0/internal/transport/http2_client.go:0",
              "Name": "google.golang.org/grpc/internal/transport.(*http2Client).handlePing",
              "SystemName": "google.golang.org/grpc/internal/transport.(*http2Client).handlePing",
              "File": "/root/go/pkg/mod/google.golang.org/grpc@v1.34.0/internal/transport/http2_client.go",
              "Line": 0,
              "Col": 0,
              "SelfWeight": 10000000,
              "TotalWeight": 50000000
            },
            "Children": [
              {
                "0": {
                  "Frame": {
                    "Children": [
                      {"SelfWeight": 0, "TotalWeight": 10000000, "Frozen": false}
                    ]
                  }
                },
                "1": {
                  "Frame": {
                    "Key": "runtime.newobject:/usr/local/go/src/runtime/malloc.go:0",
                    "Name": "runtime.newobject",
                    "SystemName": "runtime.newobject",
                    "File": "/usr/local/go/src/runtime/malloc.go",
                    "Line": 0,
                    "Col": 0,
                    "SelfWeight": 490000000,
                    "TotalWeight": 3610000000
                  }
                }
              ]
            }
          }
        }
      ]
    ]
  }
}

```

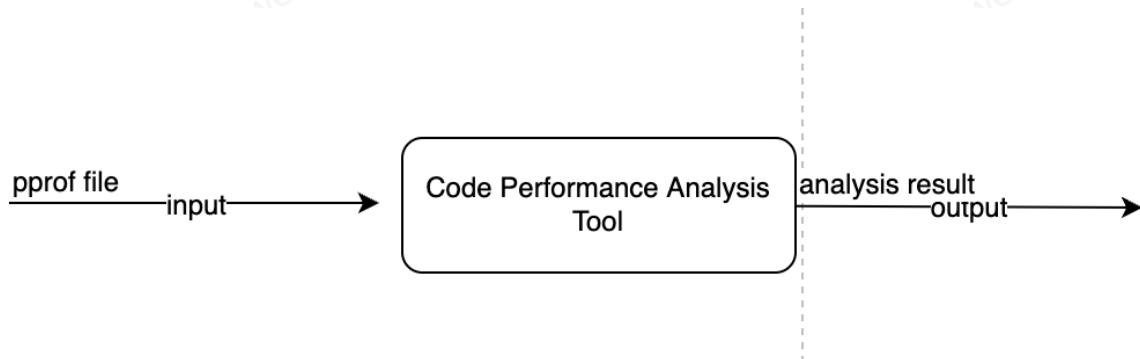
代码性能分析用例及算法：

case	key words	Judgment algorithm 1	Judgment algorithm 2
反射	reflect.*、reflect.New、reflect.Value*	包含关键词的方法占比较高(8%)	
序列化/反序列化	json.Marshal、json.Unmarshal、gob.(*Encoder).Encode、gob(*Decoder).Decode、msgpack.Marshal、msgpack.Unmarshal	包含关键词的方法占比较高(8%)	
字符串拼接	runtime.concatstrings、runtime.concatstrings*、fmt.Sprintf*	包含关键词的方法占比较高(8%)	包含关键词，并且runtime.gcBgMarkWorker 也占用比较高
锁	sync.(*Mutex).Lock、sync.(*RWMutex).Lock	包含关键词的方法占比较高(8%)	
range 循环 []struct	range	某普通方法占比较高，并且代码行出现关键词 range	
...			

分析报告形式和所有的稳定性分析报告一样，在此不详细说明。

功能定位：

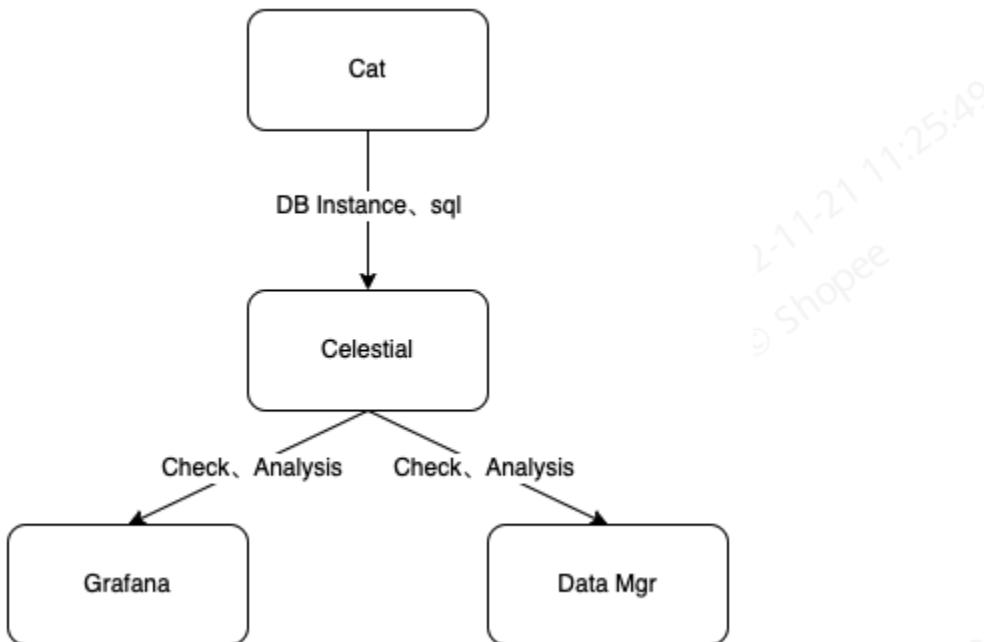
该模块的输入为pprof文件，输出prof文件分析结果。它对其他项目、环境或是中间件几乎没有依赖的，可以作为一个独立的插件让其他业务去集成。在大促系统中，计划配合日常压测，做到每日检查。



db分析

db分析包括sql和codis命令分析，下面具体介绍如何分析：

方案一：

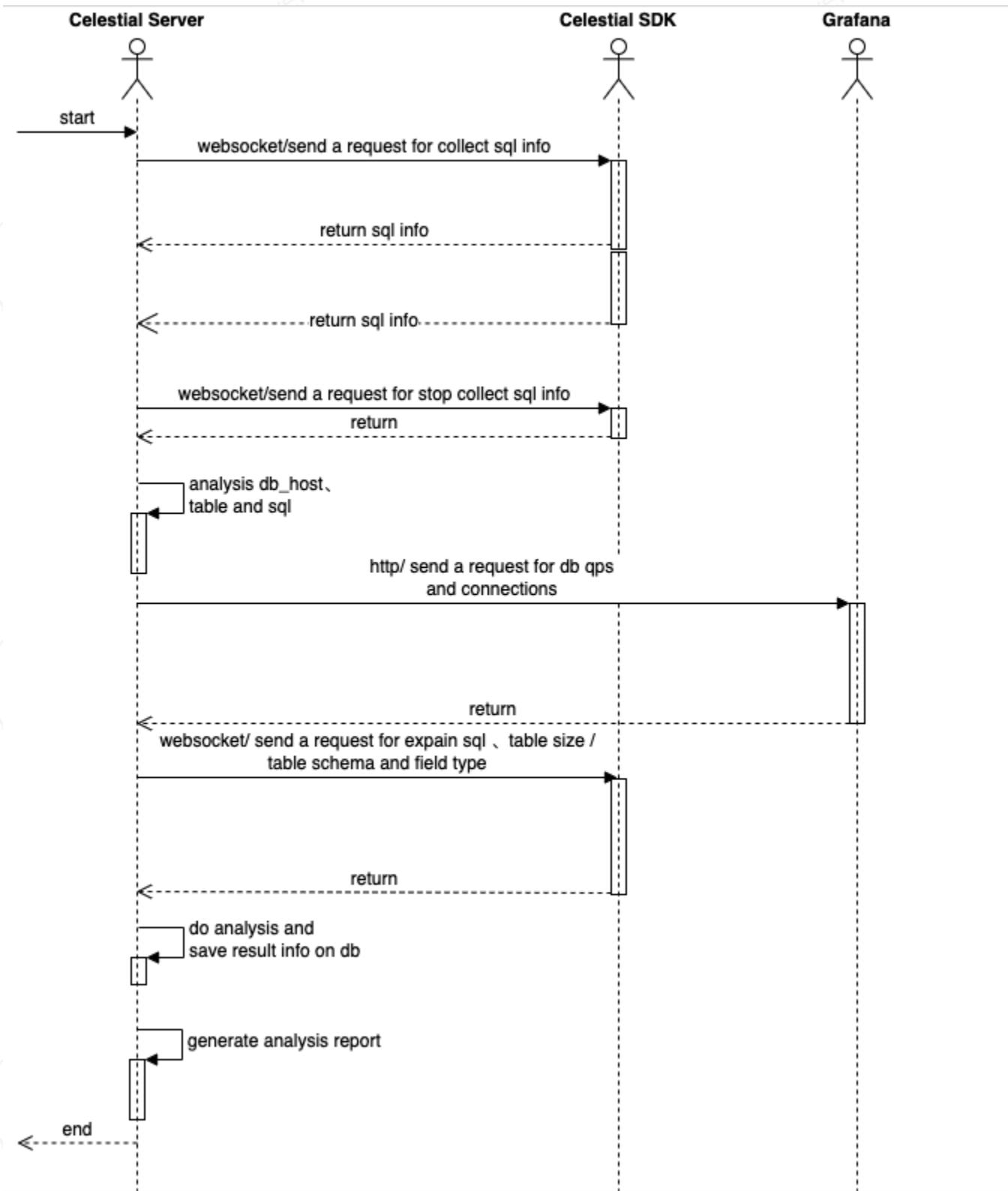


通过过cat 收集服务的数据库相关信息（如，表、查询sql等），借助promethues 的监控指标和 Data mgr的数据库查询接口进行分析。

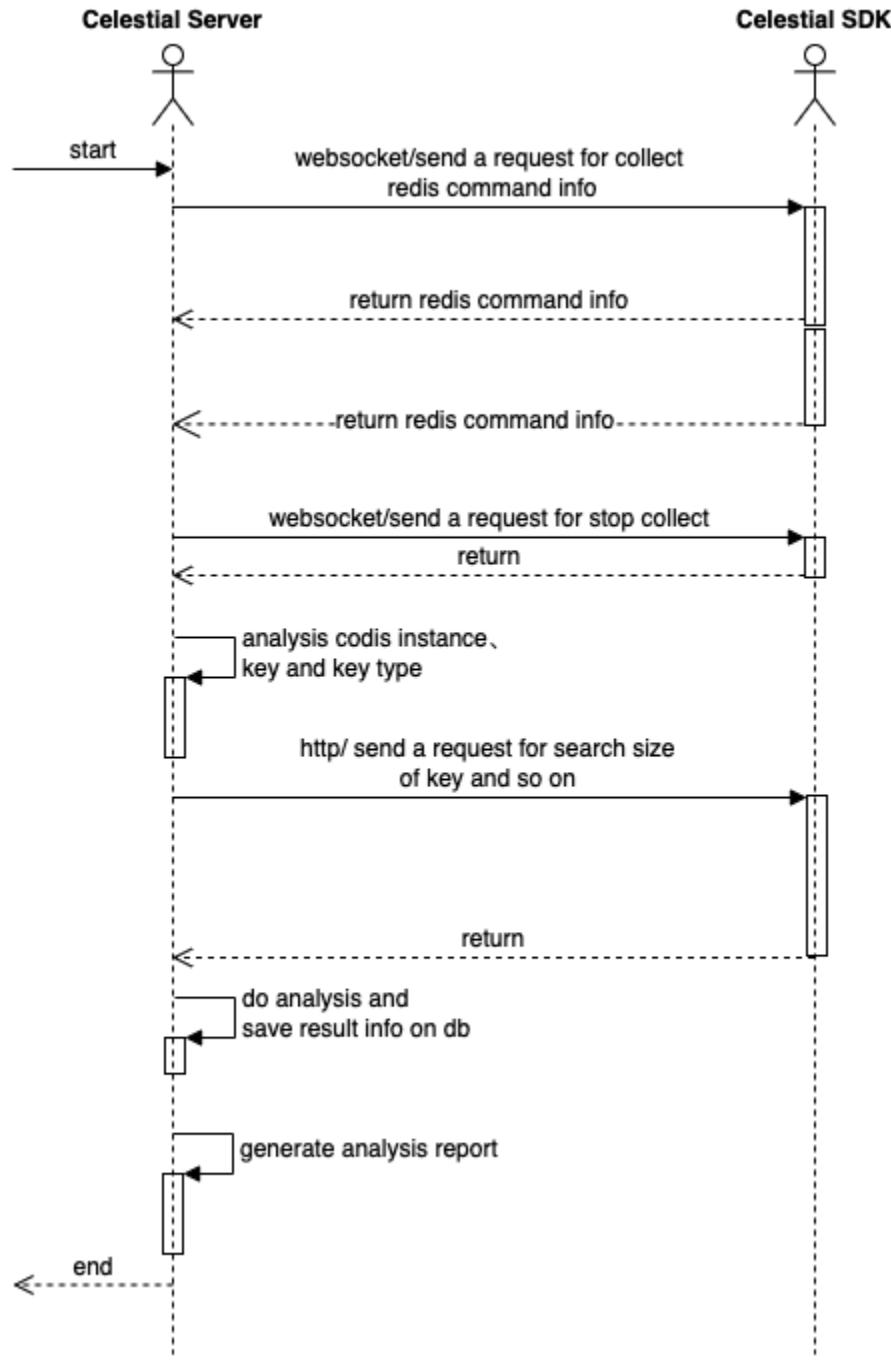
方案二：

sql信息的收集使用chassis handler机制实现，sdk内置statistics-handler，在chassis内配置启动该handler实现sql采集。handler机制同下文混沌工程。

mysql db分析时序图



大促系统从SDK收集到各服务的数据库及数据表之后，可以从 promethues 的 grafana 监控平台提供的接口查询读写qps和数据库连接数；可以向大促SDK发送请求，通过SDK内置的原生SQL查询方法获取数据表原信息或是sql的执行计划信息。



大促系统SDK内置redis 检查所需的命令(和mysql的情况类似，在此不多陈述)。通过大促系统触发SDK检查，并返回信息到大促系统，由大促系统分析。

分析用例及算法

mysql:

case	Judgment algorithm	remark
主键类型不合理	自增 ID 超过 int 上限	select * from information_schema.columns where table_name ='conf_tab'
单表字段过多	字段数量超过100个	select count(*) from information_schema.columns where table_name ='conf_tab'
单表数据过大	数据量超10 million	select table_rows from information_schema.tables where table_name = 'conf_tab'

单库qps较高	读qps 超 10k	query form grafana
单库连接数过高	连接数超 15k	query form grafana
...	...	

redis:

case	Judgment algorithm	remark
单个value过大	超 10k	获取相应key 的 value 然后计算其长度
hash set zset list 元素过多	超 5000	获取相应key 的 长度查询命令获取, 如list 的 llen key , set 的 scard key, hash 的 hlen key 等
...	...	

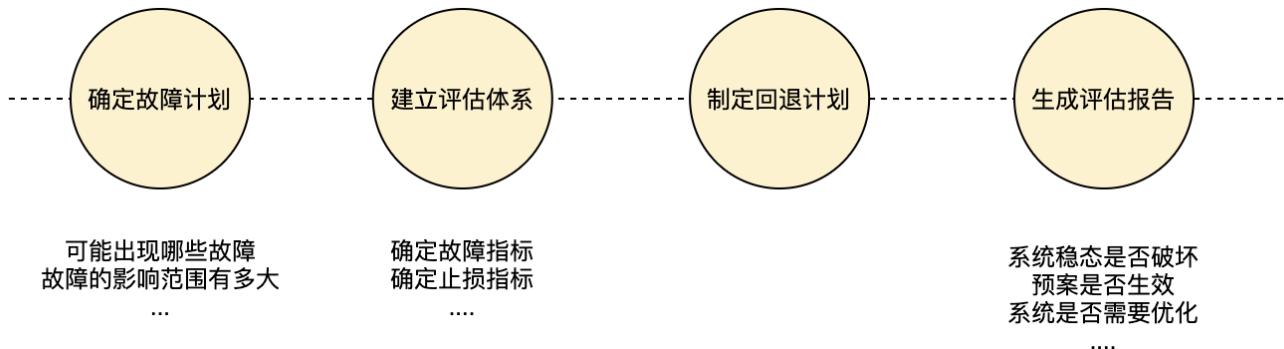
db分析的方案一是利用其他现在工具的能力来达到数据收集和查询，与方案二相比，少了对目标服务的侵入，前提是data mgr 和 devops 能够提供所需的接口或监控指标。

功能定位：

该功能对cat、grafana和data mgr 都有依赖，在大促系统中使用，可进行每日检查。

混沌工程

混沌工程通过实施详细的故障注入计划，根据系统的表现，确定优化方案、验证应急预案的一种系统稳定性保障手段，流程如下：

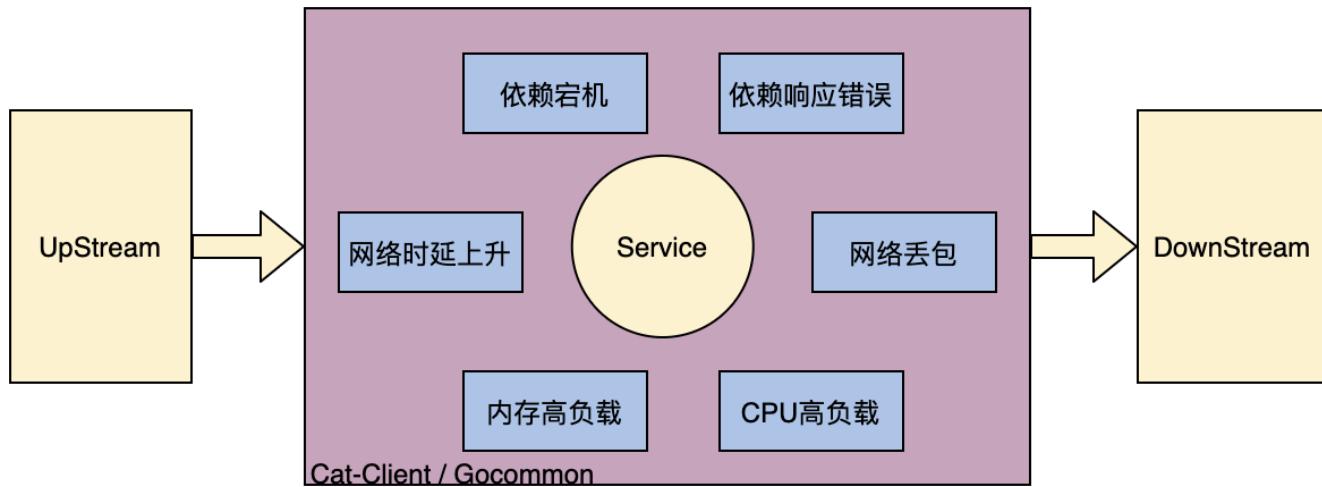


评估体系中涉及的关键指标分类如下：

关键指标	描述
故障指标	确定故障是否注入成功的直接指标，可以使用户直观的观察到故障的开始和结束，配置在故障属性中
止损指标	系统关键指标/业务核心指标所能承受的最大损失范围，到达阈值时需要立刻终止故障，确保系统不会因此产生过大的损失。 包括但不限于：关键业务指标、关键服务的系统负载、延时和错误率
关联指标	关注故障可能引入哪些关联异常特征，用于发现未知问题。例如：分布式系统本身的四项黄金指标 (latency, qps, error, saturation) 以及故障可能影响的相关服务/中间件的关键指标。

混沌工程包含一系列故障用例，通过大促SDK实现故障注入(调研是否可以复用CAT的上报埋点，改造cat-client/gocommon实现)。

通用故障用例如下：

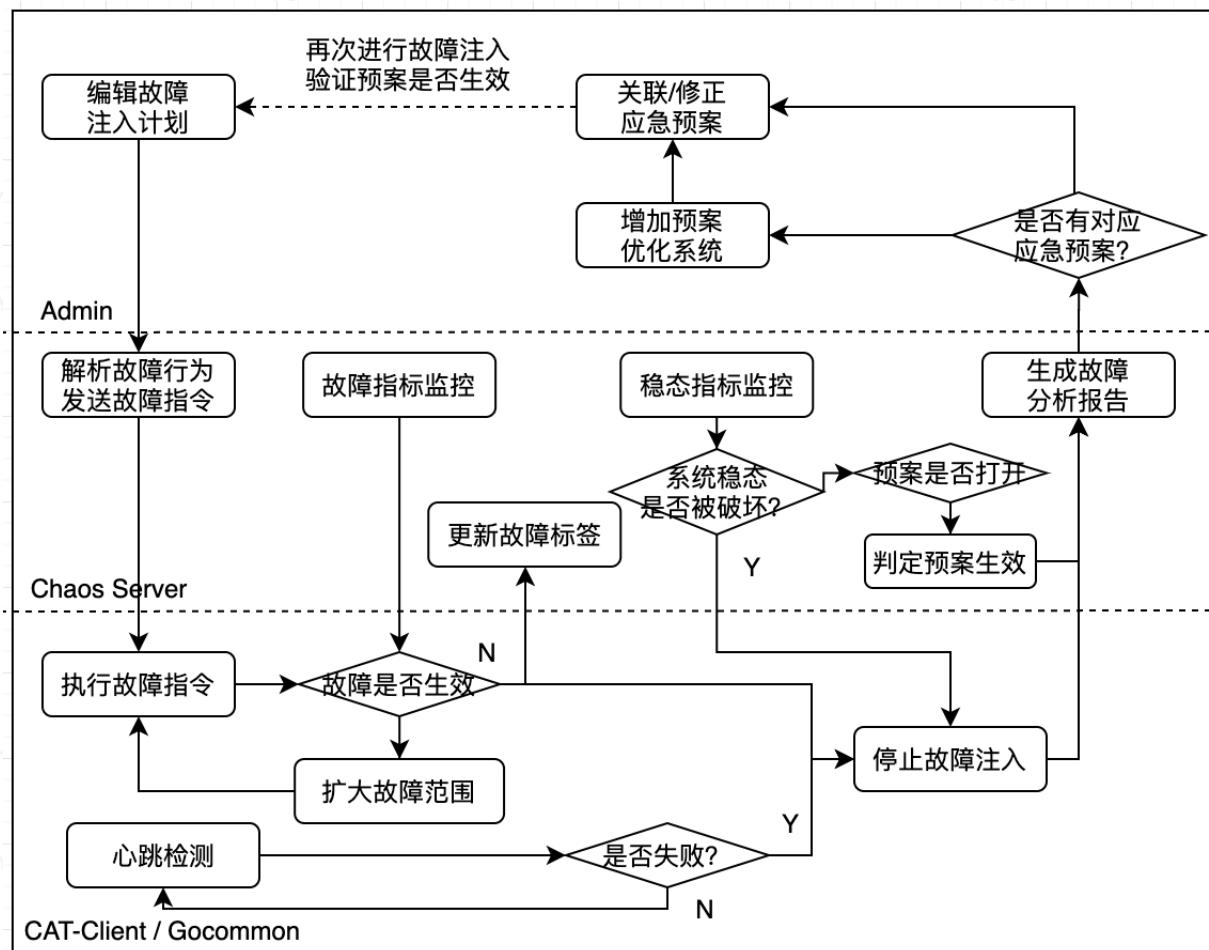


故障抽象

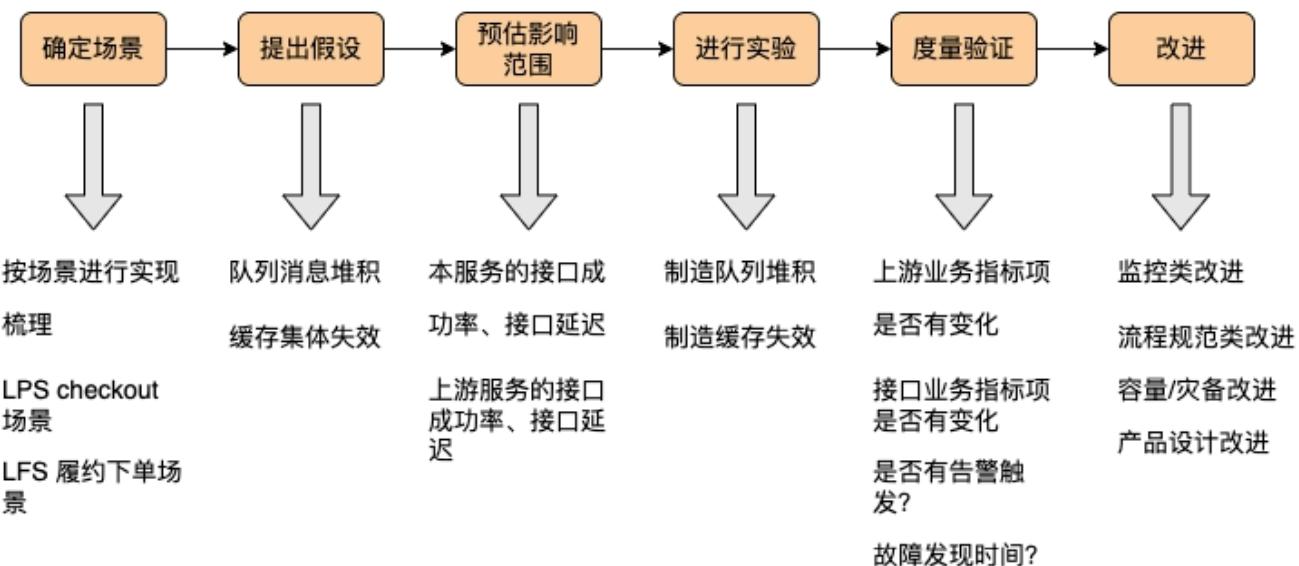
应用类	存储类	基础设施类	消息中间件
应用下线	数据库宕机	高IO	消息堆积
接口限流	连接池占满	高CPU	消息重复
内存占满	慢查询	网络延迟	消息丢失
...	数据库热点	网络丢包	...

故障类别	任务	描述	备注
基础设施类	cpu高负载	cat-client/gocommon 占用cpu	确定服务可用时的负载极限，验证相关应急预案 验证分布式系统对于网络抖动的鲁棒性
	mem高负载	cat-client/gocommon 占用mem	
	网络延时上升	cat-client中的请求装饰器/gocommon中增加调用阻塞时间	
	网络丢包	cat-client中的请求装饰器/gocommon中mock丢包	
	...		
应用类	应用响应错误	cat-client中的请求装饰器/gocommon中mock指定错误响应返回，通过configcenter维护故障注入配置信息	确定核心链路上的强弱依赖，验证降级方案/应急预案是否生效
	应用节点宕机		
	...		
中间件故障	消息重复	...	
	消息堆积		
	...		

故障下发流程：



故障计划的实施流程：



以LFS履约下单场景和LPS的checkout 场景梳理为案例：

场景	故障假设	故障实验	影响指标	度量和验证
LFS 履约下单	异步下单消息队列消息堆积	暂停异步下单的saturn任务，造成消息堆积 or 阻塞消费服务消费，延长消费时间，造成消息堆积	/api/sls/order/create 接口错误率、延迟； LFS 服务cpu、内存、LPS服务cpu、内存等	指标是否有变化 是否存在指标监控 故障发现时间花费多久 是否存在预案 预案是否有效
	异步下单消息队列消息丢失	在下单消息发送到队列时拦截丢弃或消费到消息时拦截丢弃		
	3PL下单失败	调 lcs-api /api/lts/create_line_waybill 接口下单时返回失败	/api/sls/order/create 接口错误率、延迟； LFS 服务cpu、内存、LPS服务cpu、内存； 队列堆积等	
	...			
	LPS Checkout	依赖服务接口延迟增加 redis缓存失效 ...	checkout 接口的错误率、延迟等	

混沌工程实现“三可”：

安全可控	可测可验	真实可信
1.采用xx环境来模拟live环境，不影响生产 2.混沌工具client运行增加权限，指定授权环境方可运行，避免误部署到其他环境制造破坏 3.每个故障注入都有相应的终止操作和熔断操作，来控制“最小爆炸半径” 4.若利用同步live数据做为实验数据的话，给数据添加xx的标识，以防数据竟然意外流到其他live服务无法识别 ...	1.每一项故障注入都可实现 2.故障注入效果可监测，故障下发生情况可跟踪到效果 3.每一项稳态指标都有相应的监控或告警 ...	1.实验环境真实性，部署环境和live使用相同类型容器、相同分支代码等 2.配置信息真实性，apollo相关配置信息同步live（？如何同步，还是修改代码直接识live数据） 3.业务数据真实性，SLS以外服务使用mock实现（？直接同步live数据） ...

混沌工程分析报告

服务	故障行为	优先级	是否注入成功	用例是否通过	关联预案	预案描述	预案是否生效	影响范围	备注
SLS-LIVETESTAPI-LIVE-ID	LCOS调用时延上涨10ms	P1							故障未破坏系统稳态，用例通过
LFS-LIVETESTGRPC-LIVE-ID	AWB报错率上涨10%	P2			O-LFS-32	生成面单接口限流		阻塞打印面单，阻塞卖家发货	故障会破坏系统稳态，预案生效，用例通过
LPS-LIVETESTAPI-LIVE-ID	SpexService 服务故障	P1			???	???		阻塞加入购物车/checkout	无对应应急预案，待补充
...	...								

稳定性分析报告

上述用例逐一执行完毕，汇总执行结果形成分析报告。

静态分析报告

服务	代码性能分析				DB性能分析		Codis性能分析	
	反射	序列化	锁	字符串拼接	主键类型不合理	链接数太多	存在大key	存在数据热点
SLS-LIVETESTAPI-LIVE-ID	⚠							
LPS-LIVETESTAPI-LIVE-ID	⚠	⚠						
LFS-LIVETESTGRPC-LIVE-MY							⚠	

详情表

服务	性能问题	分析原因
SLS-LIVETESTAPI-LIVE-ID	存在代码性能问题	反射使用耗时占比20%，代码调用链为：xxxxxxxx
LPS-LIVETESTAPI-LIVE-ID	存在DB性能问题	xxxxxx

表示分析结果在安全区域

表示分析结果在危险区域，需要排查修改

⚠ 表示分析结果在临界区域，需要注意或排查

代码分析和DB分析的各分析项都会有两个值：安全值和危险值。这两个值将结果划分为三个区域(当安全值和危险值相等时就是两个区域)

save point



打分机制：

1.评分的维度是服务并且区分地区(如：LPS-LIVETESTAPI-LIVE-ID、LFS-LIVETESTRPC-LIVE-MY)，总分100，60为及格线。

2.先将100平分到每一项，如总共有10项那么每项即10分，若有20项，每项即5分。

3.假设每项10分，单项的分析结果若在安全区域，得分为单项的满分，即10分；若在临界区域，得分为单项的及格分，即6分；若在危险区域，得分为0分。

4.最后总分为各项的得分之和。最后可设置一个优秀结果：没有并且在80%以上的可设置为优秀。

按照上述打分机制，上述表格的得分分别为：

服务	得分	备注
SLS-LIVETESTAPI-LIVE-ID	95	优秀
LPS-LIVETESTAPI-LIVE-ID	77.5	及格
LFS-LIVETESTRPC-LIVE-MY	57.5	不及格

非功能特性设计

可靠性

故障注入的可靠性由db可靠性和报错用户重试保证

故障下发到client的可靠性由websocket重连机制保证

子模块异常时无条件终止实验，包括：

- chaos server异常：client ping失败
- chaos db 异常：notifyService访问db失败，或读取数据解析失败
- 监控异常：monitorService访问监控失败，或读取数据解析失败，或数据值超出阈值

可运维

chaos server嵌入在大促系统内，随大促系统部署运维

chaos client嵌入在cat-client内，随应用服务部署运维，Client状态配置相应的监控面板

安全性

1. 代码分析与服务的交互只有生成pprof文件，在live或是非live环境操作对系统都几乎没有影响(不要在大促期间的live操作)，不影响系统安全。在分析结果报告中，会将代码的路径作隐藏处理，只展示关键类和方法。
2. DB分析执行的大部分是查询操作，在非大促期间一般不会对系统造成影响。在执行explain sql时，需要注意对原sql的真实用户信息作隐藏，最好能用模拟数据替换。
3. 混沌工程的落地先在test环境做测试，而后向livetest环境-live环境演进。
4. non-live测试流量使用全链路压测平台对LIVE流量进行流量回放，向live环境推进时需要模拟创造实验流量，改进cat-client可以支持过滤实验流量。
5. 为了可能的减少对业务代码的侵入，混沌工程采取改写cat-client的方式进行故障注入。但是，需要对改动版本作权限管控，限制环境与服务/PodIP，限制信息配置在chaos_config中。

可测试性

可在test以及livetest测试

监控

代码分析和DB分析都在分析完成后通过seataalk或邮件发布分析报告。

配置故障注入的Grafana监控面板 [故障指标面板，关键指标面板，Client-Status面板]