Chaos Mesh®

GraphQL in Chaos Mesh

如何高效地控制集群中的 资源状态

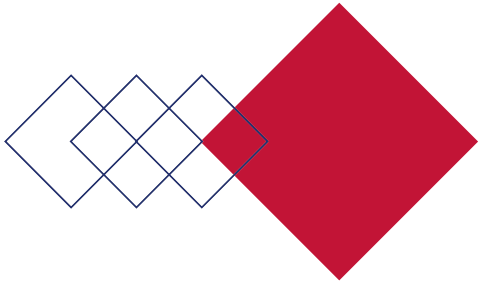
李晨曦

GitHub: hexilee
PingCAP R&D

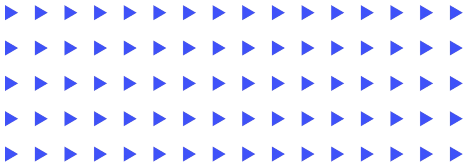
PingCAP 研发工程师，CNCF 开源项目 Chaos Mesh® 核心贡献者，主要负责工程效率提升和HTTP 故障注入功能的设计实现。并推动 GraphQL 在 Chaos Mesh 项目中的实践落地。

目录

Chaos Mesh®



Chaos Mesh 介绍



Chaos Mesh 是什么

- Kubernetes 上的云原生混沌工程平台
- 最初目标是作为 TiDB 的内部测试平台
- 提供对 Pod 或者具体容器的错误注入，包括网络、系统 IO、内核以及一些应用层注入

chaos-mesh.org

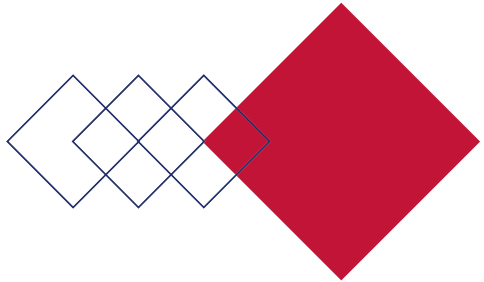
github.com/chaos-mesh

Chaos Mesh 是什么

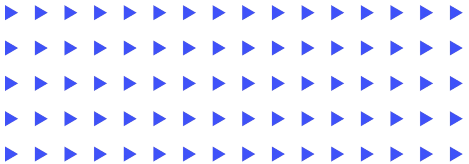
我们的目标

- 建立一个完全闭环的云原生混沌工程平台
- 让混沌工程变得更易用

Chaos Mesh®

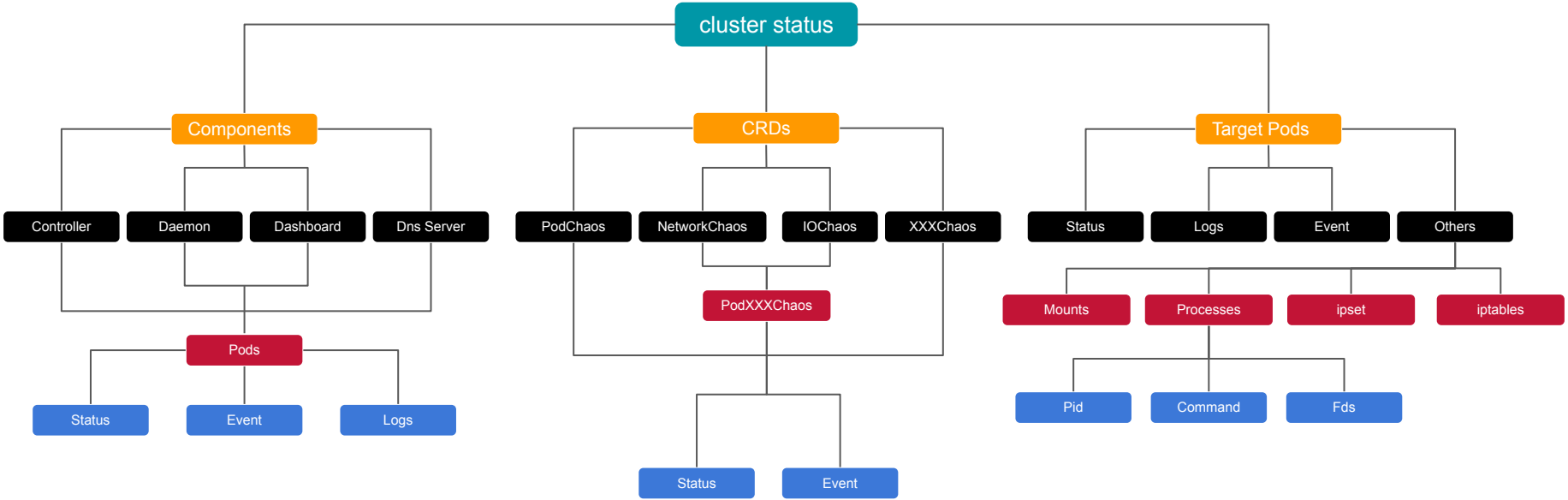


问题与解决方案



集群中的状态

Chaos Mesh 本身的运行和注入的故障会 给各组件以及目标 Pod 带来各种状态。



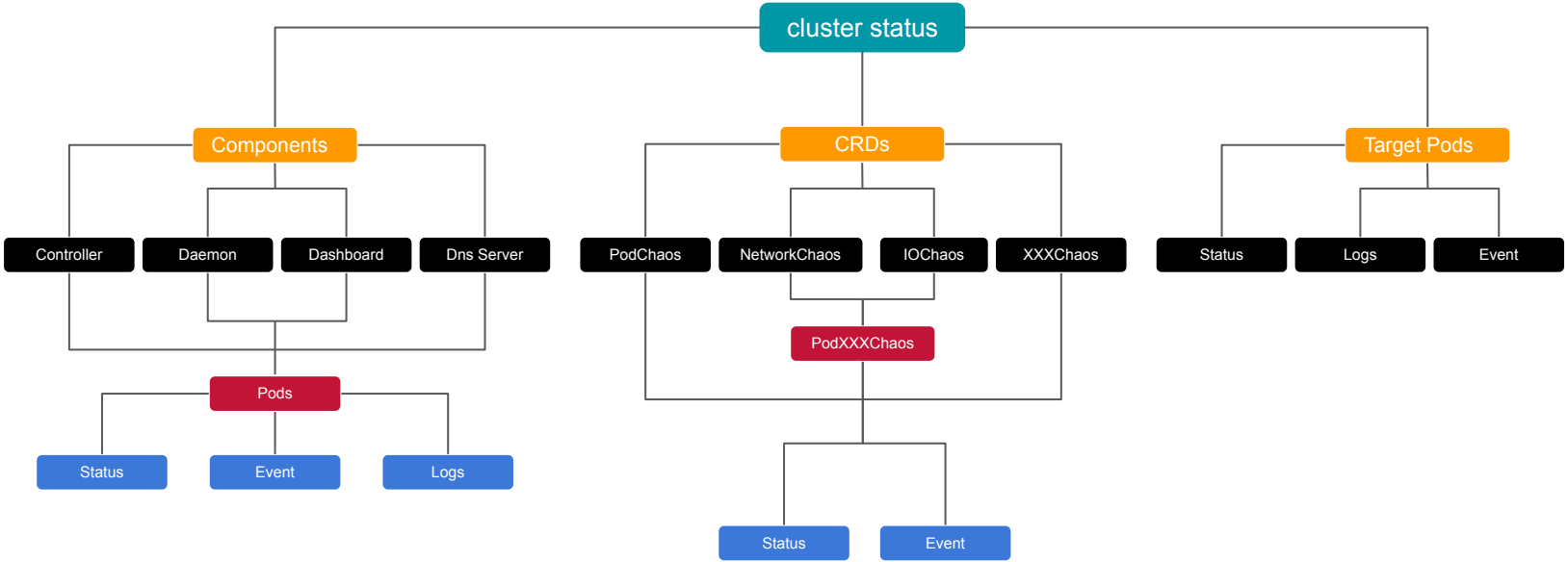
集群中的状态

在实际混沌实验过程中，可能会出现注入的错误不符合预期，甚至完全没有效果的情况。能否高效地获取各种状态则决定了故障诊断的效率。

集群状态大致可以分为两类，主要分类依据是能否通过 kubernetes API 直接查询。

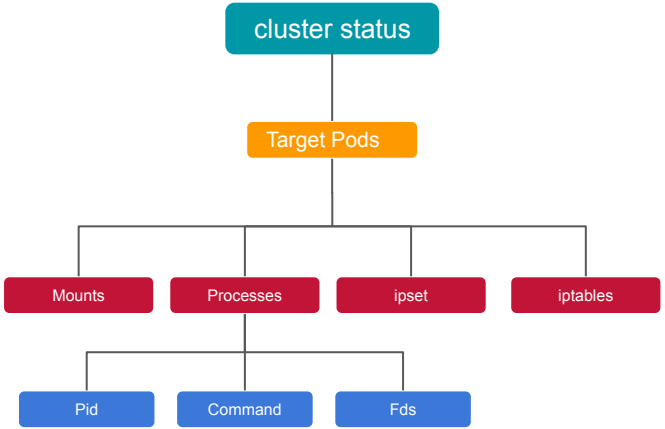
k8s 可直接查询的状态

Kubernetes 和 Chaos Mesh 组件运行的状态均可直接通过 k8s API 查询。



k8s 不可直接查询的状态

Chaos Mesh 注入的故障给目标 Pod 带来状态不可通过 k8s API 直接查询。



状态查询的障碍1

对于可通过 kubernetes API 直接查询的状态，使用过程存在的一大障碍是噪音 过多。很多情况下需要配合文本 查询工具一起使用才能找到想要的信息。

当然，kubectl 提供了 json path 功能以对查询结果进行筛选，但它存在命令行交互不友好、复 杂、难以阅读等问题。

```
hexi@Arch ~> kubectl get pods -o=jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.status.startTime}{"\n"}{end}'
nginx-5b678cd44f-v9r8m  2021-09-13T15:18:55Z
nginx-5b678cd44f-vpzvs  2021-09-13T15:18:55Z
nginx-5b678cd44f-z9r92  2021-09-13T15:18:55Z
hexi@Arch ~>
```

上图为 kubectl 的 json path 使用样例

状态查询的障碍2

对于不可通过 kubernetes API 直接查询的状态，往往要通过创建 pod/exec 子资源，运行自定义命令来获取。它存在的主要问题是查询客户端所需权限过高。

pod/exec 使用样例：列出 daemon pod 上正在运行的进程。

```
hexi@Arch ~ [1]> kubectl exec chaos-daemon-qqbwj -n chaos-testing -- ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1  93116  8968 ?        Ss   Oct20   0:40 /sbin/init noembed norestore
root           2  0.0  0.0      0     0 ?        S    Oct20   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   Oct20   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   Oct20   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   Oct20   0:00 [kworker/0:0H-kblockd]
root           8  0.0  0.0      0     0 ?        I<   Oct20   0:00 [mm_percpu_wq]
root           9  0.0  0.0      0     0 ?        S    Oct20   0:09 [ksoftirqd/0]
root          10  0.1  0.0      0     0 ?        I    Oct20   2:17 [rcu_sched]
root          11  0.0  0.0      0     0 ?        I    Oct20   0:00 [rcu_bh]
root          12  0.0  0.0      0     0 ?        S    Oct20   0:00 [migration/0]
root          13  0.0  0.0      0     0 ?        S    Oct20   0:00 [cpuhp/0]
root          15  0.0  0.0      0     0 ?        S    Oct20   0:00 [cpuhp/1]
root          16  0.0  0.0      0     0 ?        S    Oct20   0:00 [migration/1]
root          17  0.0  0.0      0     0 ?        S    Oct20   0:11 [ksoftirqd/1]
```

状态查询的障碍3

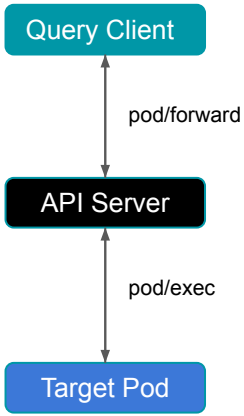
对于所有的状态查询都存在的一大问题是，各级状态之间很难进行关联查询。

人脑关联查询示例：

```
hexi@Arch ~ [1]> kubectl get httpchaos http-fault -o=jsonpath='{.status.instances}'
{"default/nginx-5b678cd44f-v9r8m":3,"default/nginx-5b678cd44f-vpzvs":3,"default/nginx-5b678cd44f-z9r92":3}
hexi@Arch ~> kubectl describe pod nginx-5b678cd44f-v9r8m
Name:         nginx-5b678cd44f-v9r8m
Namespace:    default
Priority:     0
Node:         minikube/192.168.39.176
Start Time:   Mon, 13 Sep 2021 23:18:55 +0800
Labels:       app=nginx
              pod-template-hash=5b678cd44f
Annotations:  <none>
Status:       Running
IP:           172.17.0.7
```

状态查询的解决方案

首先我们考虑障碍1，要避免查询客户端所需权限过高，最简单的办法就是在一个拥有创建 pod/exec 权限的组件上运行一个 API server 来运行查询命令，而查询客户端仅创建 pod/forward 权限即可与 API server 通信。



状态查询的解决方案

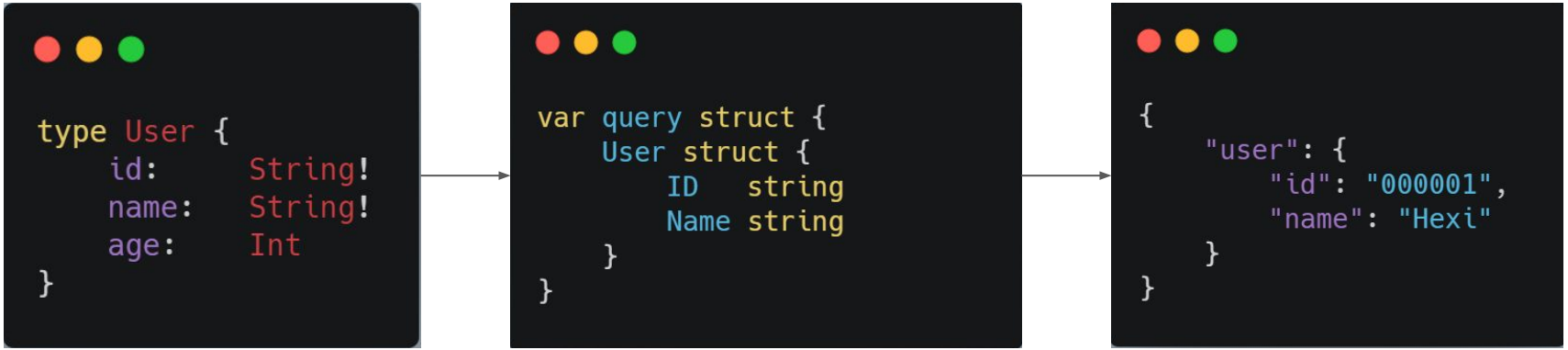
现在假定我们已经引入了一个 API server，仅考虑障碍1的查询噪音问题，有 nested resources（类似 k8s API 中的 sub resources）和 GraphQL 两种 API 方案可以选择。

但如果要解决障碍3中的关联查询问题，则 GraphQL 是最佳的 API 方案。

GraphQL

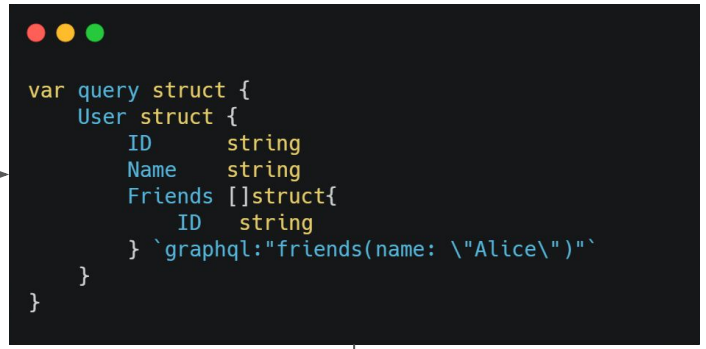
GraphQL 旨在让 API 变得快速、灵活并且为开发人员提供便利。它可让 API 维护人员灵活地添加或弃用字段，而不会影响现有查询。开发人员可以使用自己喜欢的方法来构建 API，并且 GraphQL 规范将确保它们以可预测的方式在客户端发挥作用。

GraphQL


type User {
id: String!
name: String!
age: Int
}
var query struct {
User struct {
ID string
Name string
}
}
{
"user": {
"id": "000001",
"name": "Hexi"
}
}

GraphQL


type User {
id: String!
name: String!
age: Int
friends(name: String): [User!]
}


var query struct {
User struct {
ID string
Name string
Friends []struct{
ID string
} `graphql:"friends(name: \"Alice\")"`
}
}


{
"user": {
"id": "000001",
"name": "Hexi",
"friends": [{
"id": "000002"
}]
}
}

GraphQL

Pod 与 PodIOChaos 之间的一对一关联：

```
type PodIOChaos {
    pod: Pod!
}
```

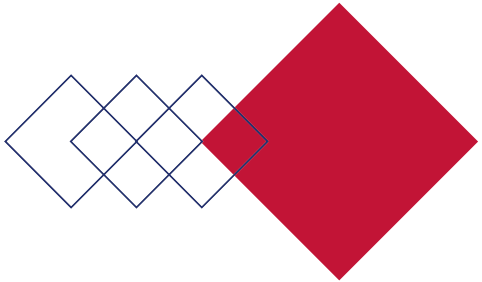
```
type Pod {
    podiochaos: PodIOChaos
}
```

IOChaos 与 PodIOChaos 之间的多对多关联：

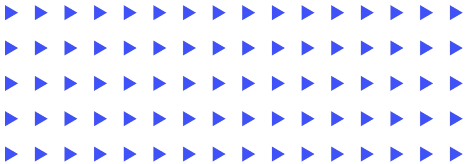
```
type PodIOChaos {
    iochaos: [IOChaos!]
}
```

```
type IOChaos {
    podiochaos: [PodIOChaos!]
}
```

Chaos Mesh®



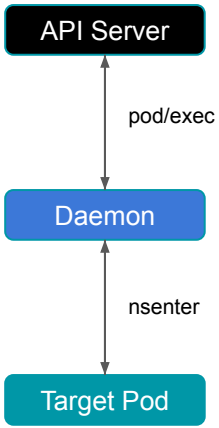
设计思路与实现



API server

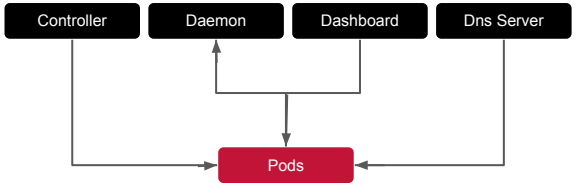
API server 最佳的载体组件还是 controller manager，因为它方便将具体的状 态查询请求分发到各 k8s node 上的 chaos daemon。

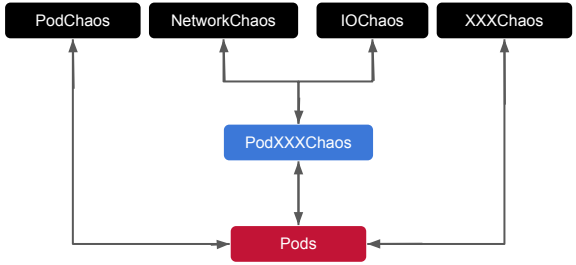
具体的做法是在 chaos daemon 的安装 namespace（如 chaos-testing）中给 controller manager 创建 pod/exec 的权限，然后创建 chaos daemon pod 上的 exec 命令，调用 nsenter 以在各节点的目标 pod 上运行查询命令。



API server

在 server 端通过 GraphQL API 将各种状态关联起来，便于查询。





Query Client

Client 主要使用 [go-graphql-client](#) 实现查询。这是一个基于反射的 GraphQL 客户端库

```
var query struct {
    Me struct {
        Name graphql.String
    }
}
```



```
query {
    me {
        name
    }
}
```

它根据结构体生成查询，并自动将查询结果反序列化给结构体实例赋值。

Query Client

我们采用全动态查询，客户端会自动解析 GraphQL API 的 scheme，并根据 query path，通过反射生成结构体。

```
chaosctl get user:000001/name
```

```
var queryType = StructOf([]StructField{
    {
        Name: "User",
        Type: StructOf([]StructField{
            {
                Name: "Name",
                Type: TypeOf(""),
            },
        }),
        Tag: StructTag(`graphql:"user(id: \"000001\")"`),
    },
})
var query = New(queryType).Interface()
```

```
var query struct {
    User struct {
        Name string
    } `graphql:"user(id: \"000001\")"`
}
```

```
user:
  name: "Hexi"
```

最后将结构体实例序列化成 yaml 格式打印。

Completion

由于查询状态有非常多的排列组合，我们依赖 GraphQL API 实现了丰富的动态自动补全。

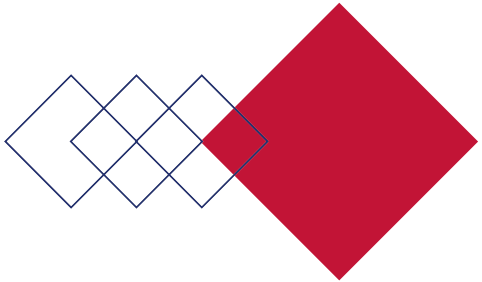
```
chaosctl get user:000001/friends:
```

```
var queryType = StructOf([]StructField{
    {
        Name: "User",
        Type: StructOf([]StructField{
            {
                Name: "Friends",
                Type: SliceOf(StructOf([]StructField{
                    {
                        Name: "Name",
                        Type: TypeOf(""),
                    },
                })),
            },
        }),
        Tag: StructTag(`graphql:"user(id: \"000001\")"`),
    },
})
```

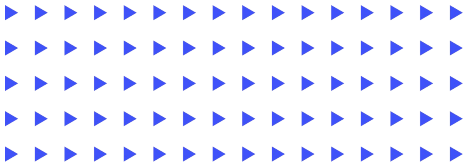
```
chaosctl get user:000001/friends:Alice
chaosctl get user:000001/friends:Bob
```

```
{
    "user": {
        "friends": [
            {
                "name": "Alice"
            },
            {
                "name": "Bob"
            }
        ]
    }
}
```

Chaos Mesh®

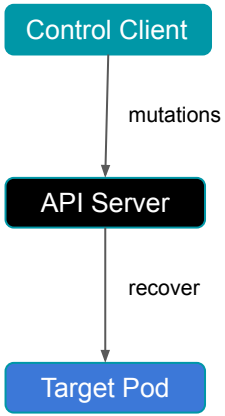


后续的工作



强制 recover

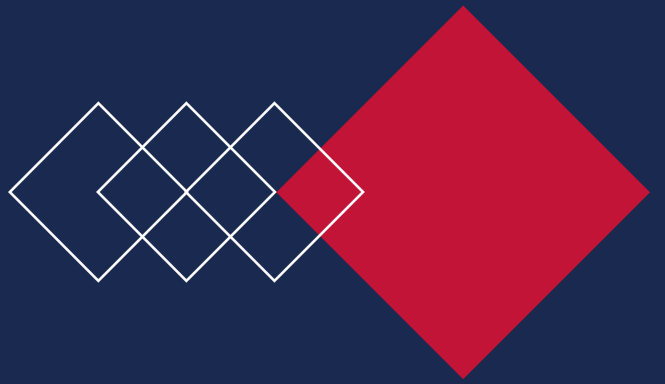
目前我们的状态控制功能都集中在 查询，而没有更改。后面最主要的工作之一就是 实现故障的强制恢复，这涉及到 daemon 状态持久化，以及 API server 需要用到 GraphQL 中的 mutations API。



命令简化

虽然我们有完备的查询路径自动补全，但很多常用状 态还是需要更方便的命令来 查询。比如可以把 chaosctl log daemon 来作为命令 chaosctl get component:daemon/logs 的 alias。这种简化命令仅需改动客户端，不需要进行 API server 的迭代更新，可以根据开 发者的使用 习惯来持续添加。

Chaos Mesh®



Thanks