Golang在工程实践中的错误处理

彭友顺

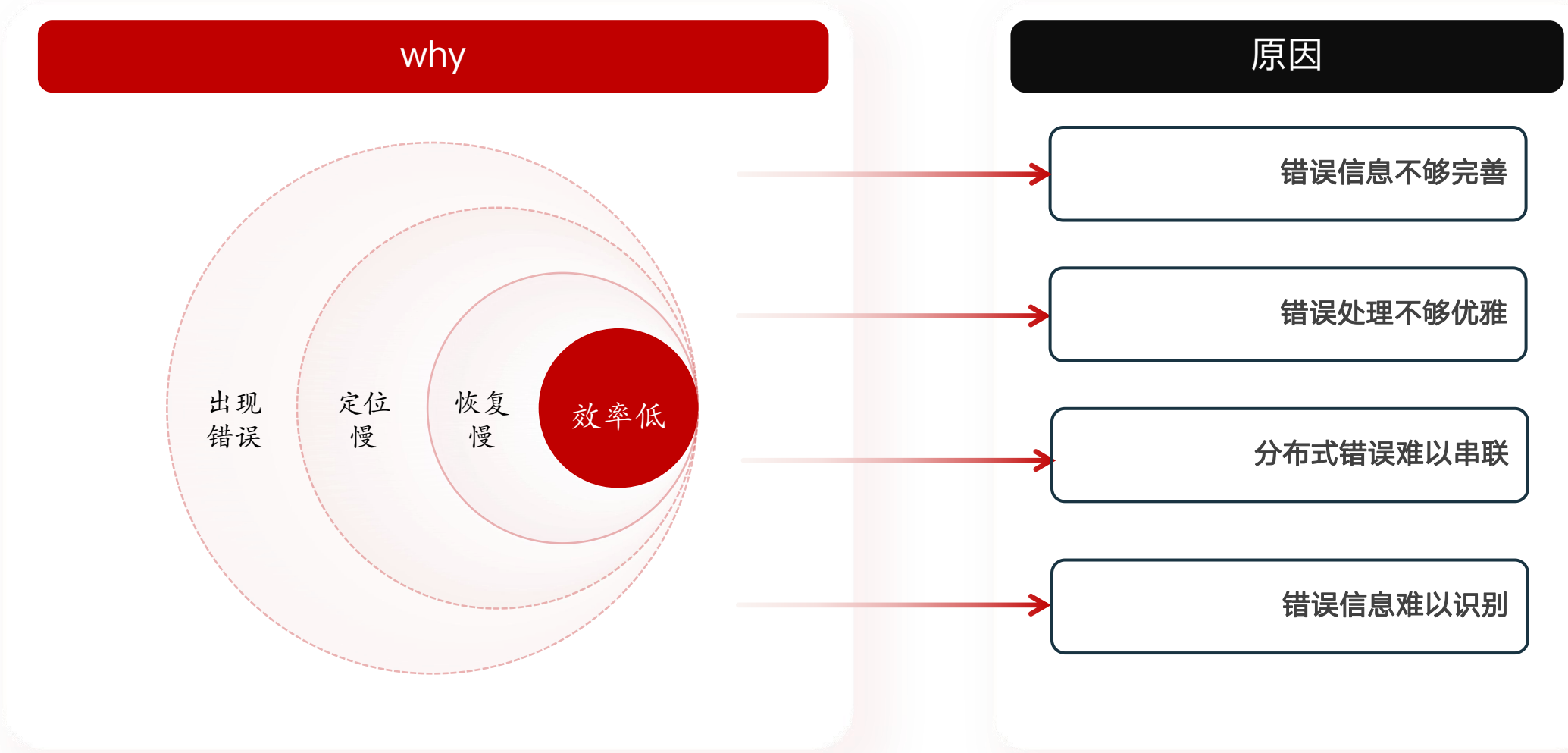
石墨文档
产研负责人

目录

第一部分

为什么我们处理错误会这么慢

为什么我们处理错误会这么慢



第二部分

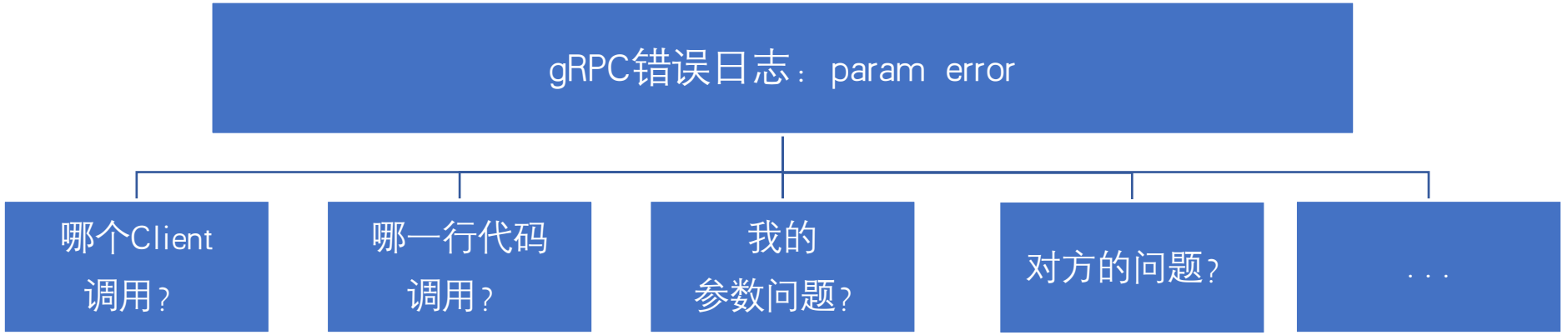
如何完善错误信息

为什么调试慢？-- 错误信息

充足信息

高亮信息

封装组件




```
{"lv":"error","ts":1711111870,"msg":"grpc error","error":"param error"}
```

遇到一个问题，我们就追加一个日志字段，排查错误效率非常低

为什么调试慢？-- 错误信息

充足信息

高亮信息

封装组件

总结起来，调试阶段需要对接的信息

对端信息	能够确定对端的唯一来源，例如对端的应用名称、对端的配置、对端的IP。
请求方法	对端请求的方法。
请求参数	请求的参数信息，包括 header 里的 metadata
响应数据	响应的数据，包括 header 里的 metadata
状态信息	错误码和错误信息
耗时时间	请求到响应的耗时时间
执行行号	调用处执行的行号

为什么调试慢？-- 错误信息

充足信息

高亮信息

封装组件

- 一堆info日志中藏着error日志
- 你能一眼找到error日志吗？


```
{"lv":"info","ts":1711111822,"msg":"access","comp":"client.egrpc","compName":"grpc.test","addr":"127.0.0.1:9001","type":"unary","code":0,"ucode":200,"desc":"OK","method":"/helloworld.Greeter/SayHello","cost":0.623,"name":"127.0.0.1:9001","tid":"ba63df5cacdc7b9356411ec8d8479b4d","req":{"metadata":{"traceparent":["00-ba63df5cacdc7b9356411ec8d8479b4d-7a84dcb4af08e5a9-00"]},"payload":"{\"name\":\"i am client\"}"},"res":{"message":"Hello EGO, I'm 0.0.0.0:9001"},"event":"normal"}
{"lv":"error","ts":1711111822,"msg":"access","comp":"client.egrpc","compName":"grpc.test","addr":"127.0.0.1:9001","type":"unary","code":14,"ucode":503,"desc":"error22222222","method":"/helloworld.Greeter/SayHello","cost":0.284,"name":"127.0.0.1:9001","tid":"e2445f555c4b803dd80e9f5db2d1b614","req":{"metadata":{"traceparent":["00-e2445f555c4b803dd80e9f5db2d1b614-f0683fda3a0837f9-00"]},"payload":"{\"name\":\"error\"}"},"res":{},"event":"error","error":"rpc error: code = Unavailable desc = error22222222"}
{"lv":"info","ts":1711111870,"msg":"read watch","comp":"core.econf","comp":"file datasource","configFile":"/Users/askuy/code/github/gotomicro/go-engineering/chapter_process_error/chapter0/grpc_not_highlight_error/client/config.toml","realConfigFile":"/Users/askuy/code/github/gotomicro/go-engineering/chapter_process_error/chapter0/grpc_not_highlight_error/client/config.toml","fppath":"/Users/askuy/code/github/gotomicro/go-engineering/chapter_process_error/chapter0/grpc_not_highlight_error/client/config.toml"}
```


- 在调试阶段，日志用红色高亮错误
- 肉眼才能最快的定位到error日志
- 利用IDE规则，直接点开代码执行行号，跳到指定的代码位置

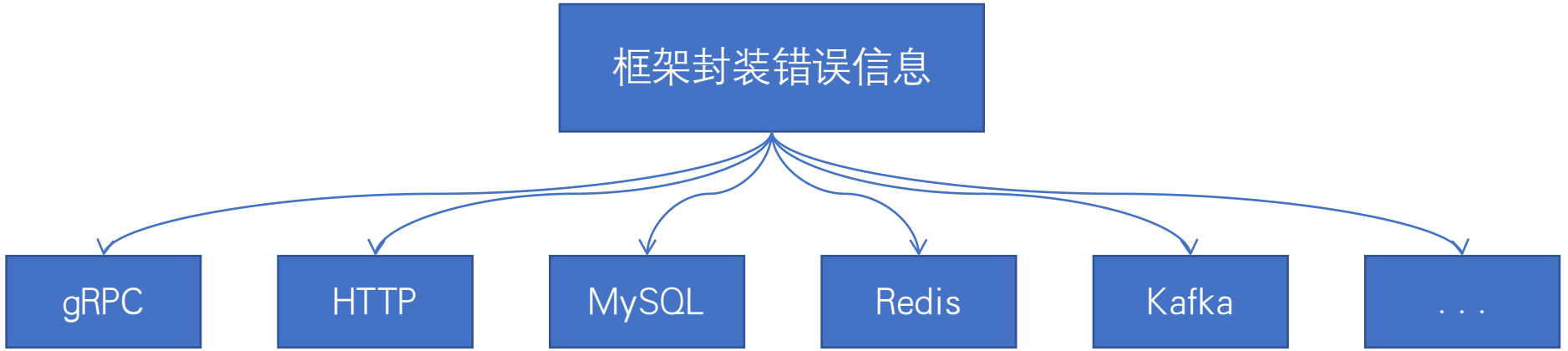
为什么调试慢？-- 错误信息

充足信息

高亮信息

封装组件

- 脏活累活交给框架
- 组件要全面统一

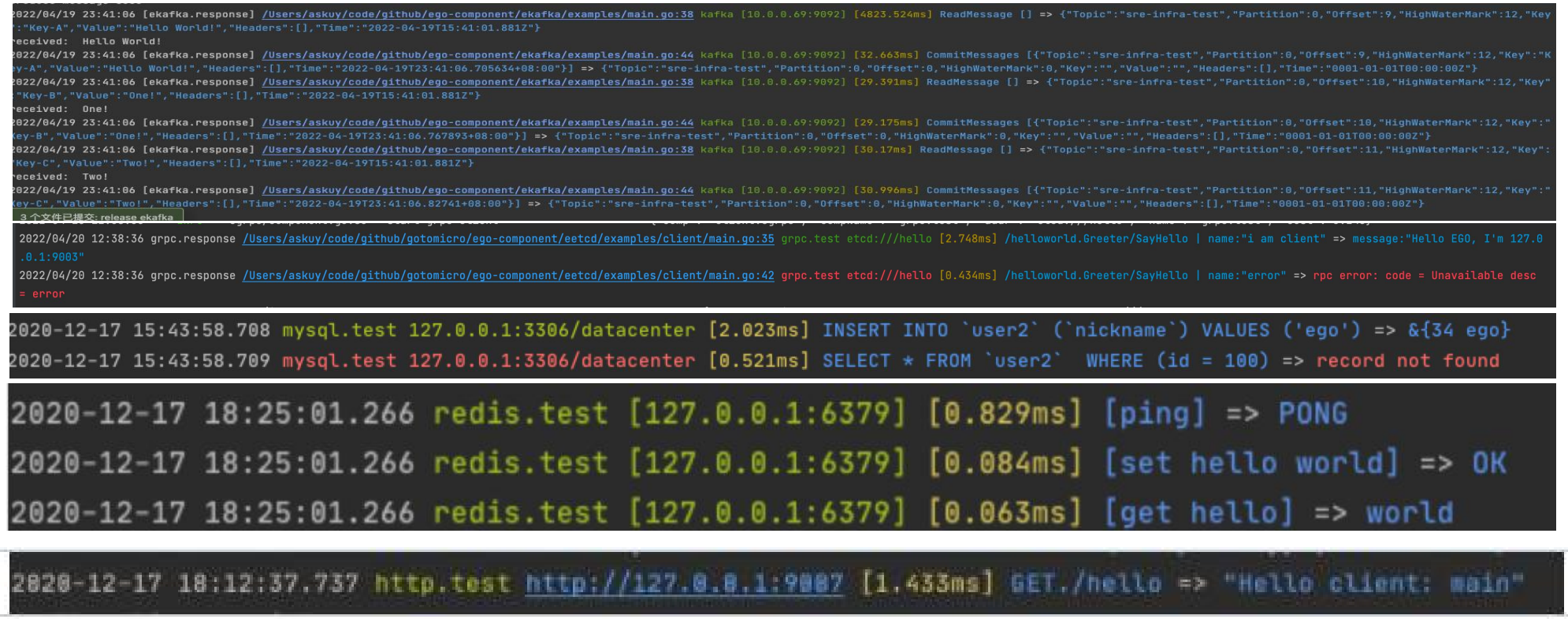


为什么调试慢？-- 错误信息

充足信息

高亮信息

封装组件



- 没有调试信息和错误信息
- 对接起来会非常麻烦

第三部分

优雅处理错误信息

为什么定位慢？-- 错误处理

记录一次错误

不要透传错误

错误码唯一性

假设用户反馈了无法打开一个文件

我们的程序员非常认真

```go
func findMysqlFile() (*File, error) {  1个用法  新*
    err := fmt.Errorf( format: "文件不存在")
    if err != nil {
        frame.LoggerError( msg: "findMysqlFile fail", zap.Error(err))
        return nil, err
    }
    return &File{}, nil
}
func findServiceFile() (*File, error) {  1个用法  新*
    file, err := findMysqlFile()
    if err != nil {
        frame.LoggerError( msg: "findServiceFile fail", zap.Error(err))
        return nil, err
    }
    return file, nil
}
func findControllerFile() {  1个用法  新*
    _, err := findServiceFile()
    if err != nil {
        frame.LoggerError( msg: "findControllerFile fail", zap.Error(err))
        return
    }
}
func main() {  新*
    findControllerFile()
}
```

记录了文件不存在的错误日志

```
ERROR   chapter1/main.go:13     findMysqlFile fail      {"error": "文件不存在"}
ERROR   chapter1/main.go:21     findServiceFile fail    {"error": "文件不存在"}
ERROR   chapter1/main.go:29     findControllerFile fail {"error": "文件不存在"}
```

- 同样的错误信息，非常多的杂音
- 每个Error，都去查看一次对应代码，排查效率低
- 占用存储空间
- 最外层入口处只记录一次错误日志

为什么定位慢？-- 错误处理

记录一次错误

不要透传错误

错误码唯一性

带来新的问题，无法定位整个代码执行链路

两个service方法都调用了findMysqlFile

日志分析不出是哪个service调用了MySQL

```go
func findMysqlFile() (*File, error) {  2 个用法  新 *
    err := fmt.Errorf( format: "文件不存在")
    if err != nil {
        return nil, err
    }
    return &File{}, nil
}
func findServiceFile1() (*File, error) {  1 个用法  新 *
    return findMysqlFile()
}
func findServiceFile2() (*File, error) {  1 个用法  新 *
    return findMysqlFile()
}
func findControllerFile(c *gin.Context) {  1 个用法  新 *
    var err error
    if c.Param( key: "type") == "1" {
        _, err = findServiceFile1()
    } else {
        _, err = findServiceFile2()
    }
    if err != nil {
        frame.LoggerError( msg: "findControllerFile fail", zap.Error(err))
        return
    }
}
func main() {  新 *
    findControllerFile(&gin.Context{})
}
```

为什么定位慢？-- 错误处理

带来新的问题，无法定位整个代码执行链路

记录一次错误

不要透传错误

错误码唯一性

- 不能透传错误，fmt.Errorf
- 如果不考虑性能
  - 日志开启Stack
  - 错误追加Stack

```go
func findServiceFile1() (*File, error) {  1个用法  新*
    file, err := findMysqlFile()
    if err != nil {
        return nil, fmt.Errorf( format: "findServiceFile1 fail, err: %w", err)
    }
    return file, nil
}
func findServiceFile2() (*File, error) {  1个用法  新*
    file, err := findMysqlFile()
    if err != nil {
        return nil, fmt.Errorf( format: "findServiceFile2 fail, err: %w", err)
    }
    return file, nil
}
```

```
ERROR   chapter4/main.go:40     findControllerFile fail {"error": "findServiceFile2 fail, err: 文件不存在"}
```

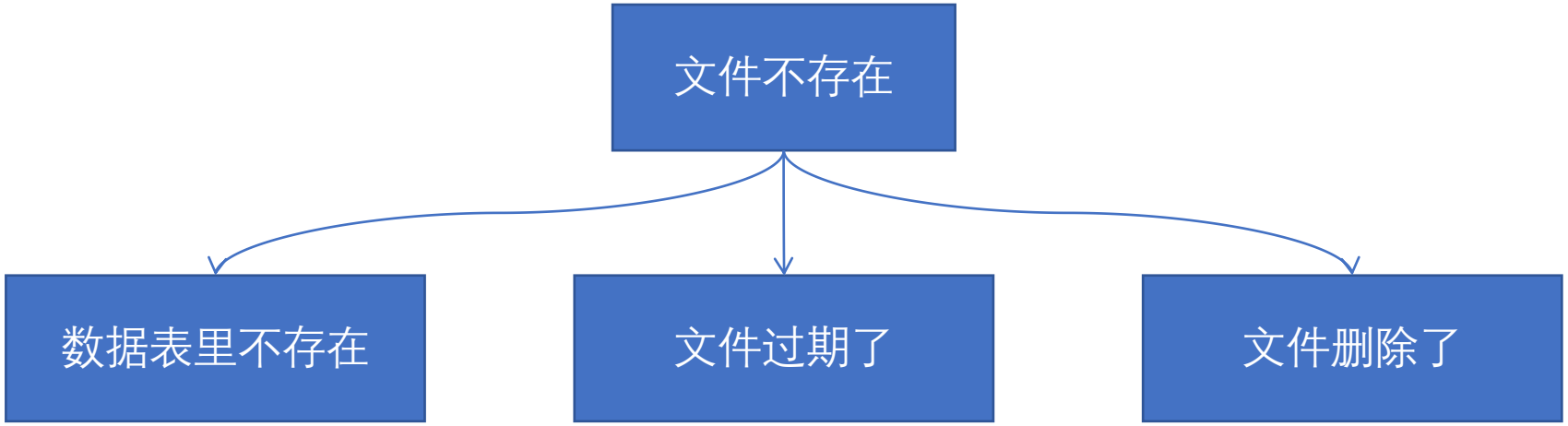
为什么定位慢？-- 错误处理

记录一次错误

不要透传错误

错误码唯一性

并没有定位到根本问题



- 一个问题，多种原因
- 无法一次性确定问题，仍需查看代码以确定具体情况
- 错误码唯一性，准确的反映出错误的根因，才能快速定位问题

如果不在写代码的时候花时间做唯一错误码，那么只能在排查的时候花更多时间查问题

第四部分

分布式错误处理

为什么定位慢？-- 分布式错误

分布式信息

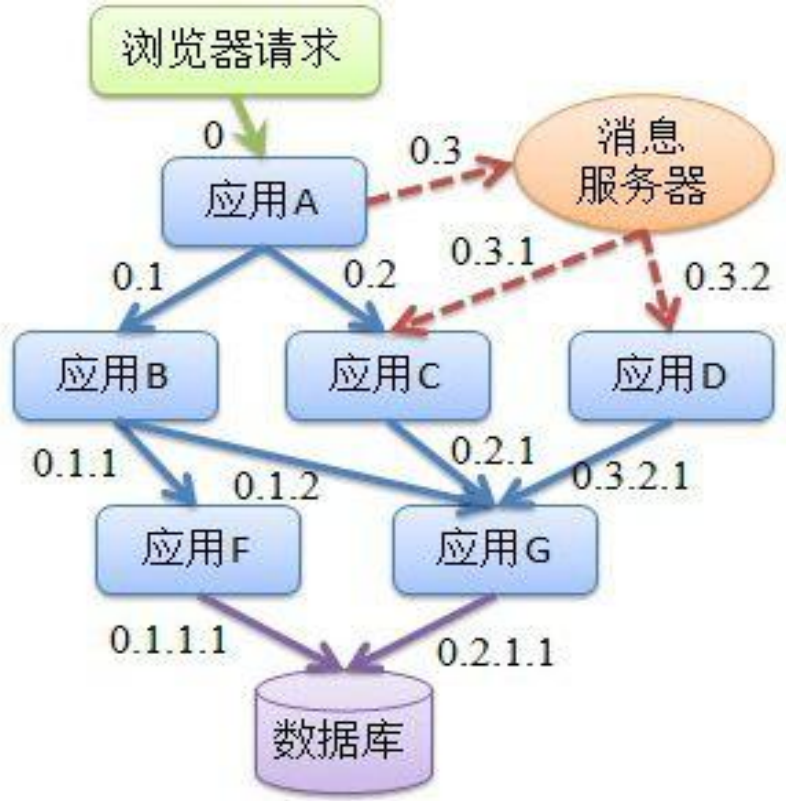
错误尽早失败

TraceId

串联信息

在微服务体系中，每个应用会涉及多种组件和调用多个业务API，导致调用链变得复杂，整体架构的复杂度也随之增加。

A服务出现了问题，可能是由其他B，C，F,G等服务引起的



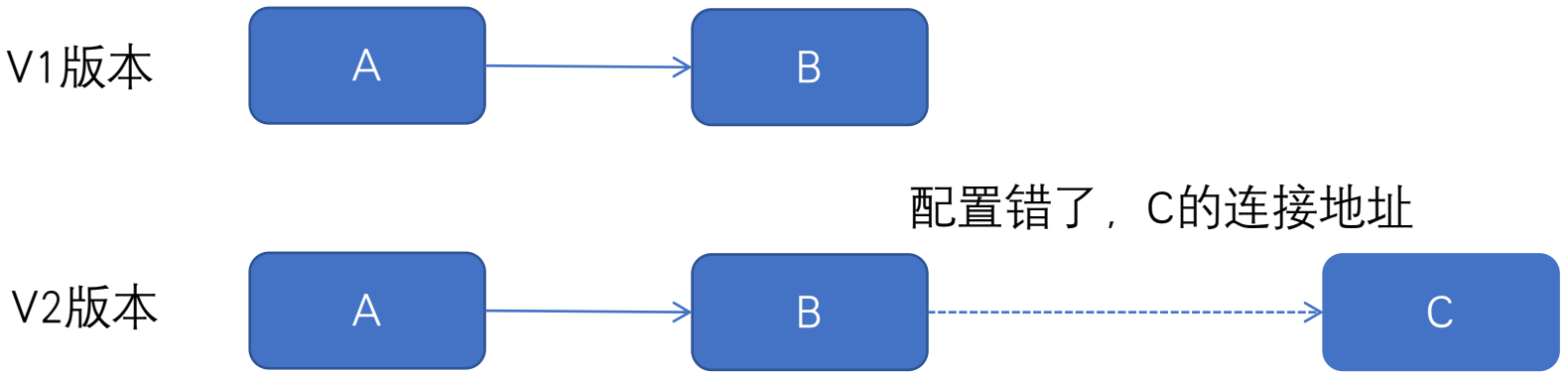
为什么定位慢？-- 分布式错误

分布式信息

错误尽早失败

TraceId

串联信息



怎么办？

- b服务记录了错误日志，但**仍然启动**b服务，a服务用了新版本的b服务。
- b服务**停止启动**，并记录错误日志，a服务用了老版本的b服务。

这两种处理错误方式，你觉得哪种是对的？

为什么定位慢？-- 分布式错误

分布式信息

错误尽早失败

TraceId

串联信息

方式一，仍然启动b服务

- b服务核心错误信息会藏在了某些日志，如下图所示
- 引入流量后，a服务中接口因为调用b服务报错，但a服务日志里不是根因
- 这种错误被隐藏，我们排查的时候需要很久
- 影响线上功能

```
2023-01-29T17:59:42.879+0800    WARN    /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:1282      [core] [Channel #1 SubChannel #2] grpc: addrConn.createTransport failed to connect to {
  "Addr": "127.0.0.1:9100",
  "ServerName": "127.0.0.1:9100",
  "Attributes": null,
  "BalancerAttributes": null,
  "Type": 0,
  "Metadata": null
}. Err: connection error: desc = "transport: Error while dialing dial tcp 127.0.0.1:9100: connect: connection refused"
google.golang.org/grpc.(*addrConn).createTransport
        /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:1282
google.golang.org/grpc.(*addrConn).tryAllAddrs
        /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:1219
google.golang.org/grpc.(*addrConn).resetTransport
        /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:1154
google.golang.org/grpc.(*addrConn).connect
        /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:809
2023-01-29T17:59:42.880+0800    INFO    /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:1106      [core] [Channel #1 SubChannel #2] Subchannel Connectivity change to TRANSIENT_FAILURE
2023-01-29T17:59:42.880+0800    INFO    /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/internal/balancer/gracefulswitch/gracefulswitch.go:266   [core] pickfirstBalancer: UpdateSubConnState: 0x140003a0
de0, {TRANSIENT_FAILURE connection error: desc = "transport: Error while dialing dial tcp 127.0.0.1:9100: connect: connection refused"}
2023-01-29T17:59:42.880+0800    INFO    /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:418       [core] [Channel #1] Channel Connectivity change to TRANSIENT_FAILURE
2023-01-29T17:59:43.881+0800    INFO    /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:1106      [core] [Channel #1 SubChannel #2] Subchannel Connectivity change to IDLE
2023-01-29T17:59:43.881+0800    INFO    /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/internal/balancer/gracefulswitch/gracefulswitch.go:266   [core] pickfirstBalancer: UpdateSubConnState: 0x140003a0
de0, {IDLE connection error: desc = "transport: Error while dialing dial tcp 127.0.0.1:9100: connect: connection refused"}
2023-01-29T17:59:43.881+0800    INFO    /Users/askuy/go/pkg/mod/google.golang.org/grpc@v1.52.1/clientconn.go:418       [core] [Channel #1] Channel Connectivity change to IDLE
```

为什么定位慢？-- 分布式错误

分布式信息

错误尽早失败

TraceId

串联信息

方法二，b服务 Fail Fast

- b服务在初始化的时候，发现了b服务连接不上c服务，停止启动
- 最后一行可以看到报错信息，业务方可以立刻查看到核心错误
- 因为没有启动成功，不会更新b服务版本
- a服务不受影响，不影响线上功能

```
panic:
    msg: 连接错误
    loc: /Users/askuy/code/geektime/gomicro/chapter2/example2/panic/client/main.go:15
    component: grpcClient
    target: passthrough:///127.0.0.1:9100
    error: connection error: desc = "transport: error while dialing: dial tcp 127.0.0.1:9100: connect: connection refused"
panic: 连接错误

goroutine 1 [running]:
gomicro/chapter2/mygrpc.LoggerPanic({0x104d57028, 0xc}, {0x1400031fe08?, 0x1d?, 0x14000410dc0?})
        /Users/askuy/code/geektime/gomicro/chapter2/mygrpc/logger.go:64 +0x80
gomicro/chapter2/mygrpc.NewGRPCClient({0x104fad040, 0x14000122000}, {0x104d62836, 0x1d}, {0x1400031ff58, 0x1, 0x1})
        /Users/askuy/code/geektime/gomicro/chapter2/mygrpc/client.go:21 +0x584
main.main()
        /Users/askuy/code/geektime/gomicro/chapter2/example2/panic/client/main.go:15 +0xd8
```

为什么定位慢？-- 分布式错误

分布式信息

错误尽早失败

TraceId

串联信息

Fail Fast

发现核心错误后，应该让系统直接Panic。

- 必要的错误信息，让我们知道是在哪个地方报错，哪个微服务有问题。
- 错误信息高亮，显示红色让错误更加明显。
- 核心错误中断系统启动，让错误显示在最后一行。
- 不让报错传递到别的服务

为什么定位慢？-- 分布式错误

分布式信息

错误尽早失败

TraceId

串联信息

击鼓传花

a服务发现报错 ——> 查看a日志——> b服务负责人

查看b日志——> c服务负责人

查看c日志——> d服务负责人

. . .

- 跨部门，跨负责人，排查问题是效率非常低
- TraceId 协同所有业务

为什么定位慢？-- 错误处理

分布式信息

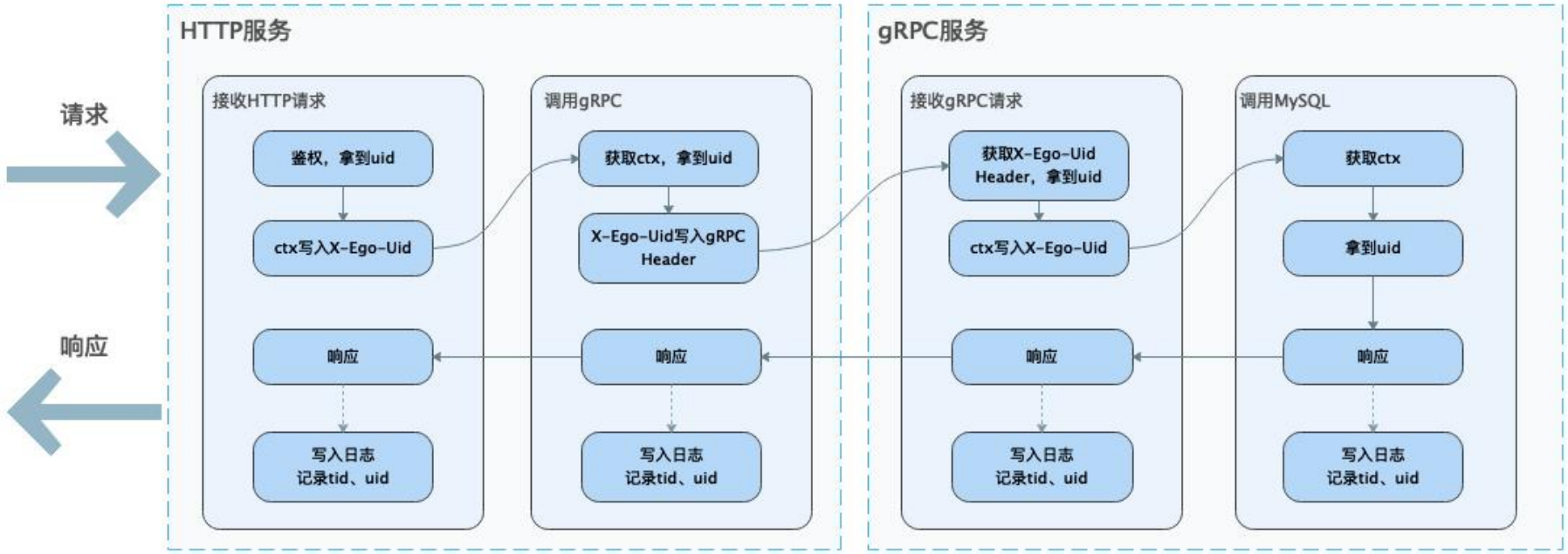
错误尽早失败

TraceId

串联信息

用户反馈的时候没有TraceId？怎么串联信息？

- UserID　跟踪用户
- TeamID　跟踪团队
- FileID　跟踪文件




```
2021-07-28 22:54:12    WARN    egorm/interceptor.go:101        access          {"comp": "component.egorm", "compName": "mysql.test", "addr": "127.0.0.1:3306", "method": "gorm:query", "name": "test.user",
"cost": 0.493, "req": "SELECT * FROM `user` WHERE id = 100 ORDER BY `user`.`id` LIMIT 1", "res": {"id":0,"name":""}, "tid": "760cfd586623e6a8", "x-ego-uid": "9527", "event": "error", "error": "record not found"}
```

为什么定位慢？-- 错误处理

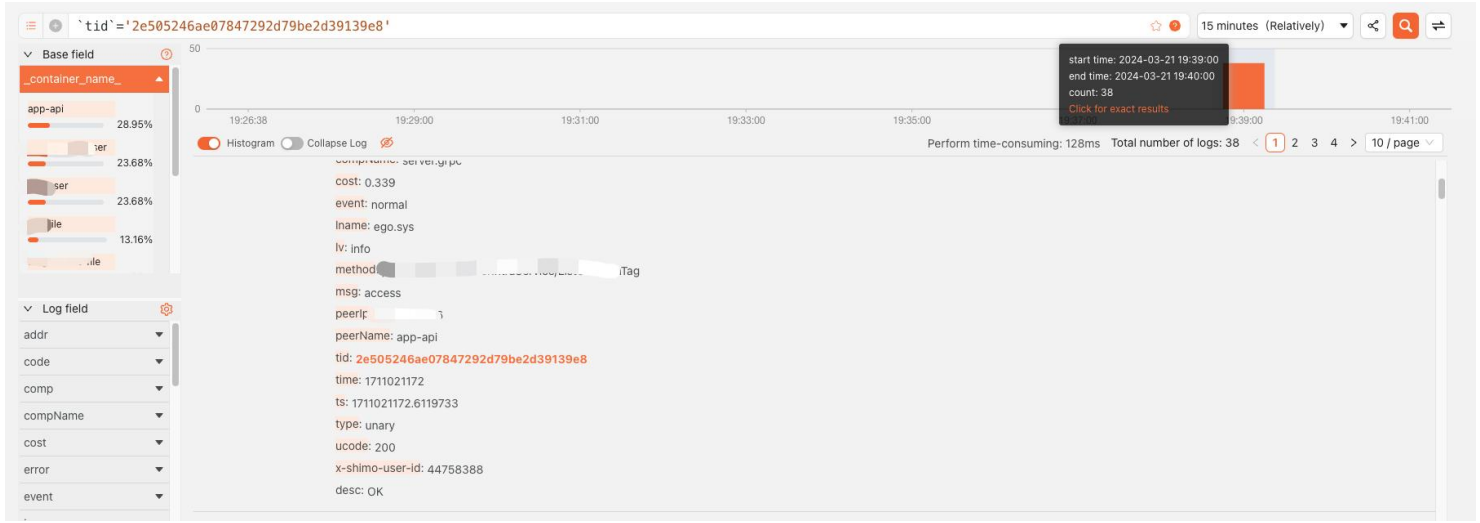
分布式信息

错误尽早失败

TraceId

串联信息

- 先Uid
- 再看Error
- 最后TraceId



第五部分

错误信息手册的必要性

为什么恢复慢？-- 错误手册

经验

识别

自动化

为什么年纪大的程序员会比年轻程序员排查问题更快

- 遇到的问题多
- 这些问题不会系统的出现在课本上
- 没遇到过的问题，很难意识到对应的操作手段

花很多时间，去排查线上问题，这是一个认知问题

为什么恢复慢？-- 错误手册

经验

识别

自动化

1. 将认知总结为文档
2. 故障演练

CODE_API_CREATE_FILE_TYPE_UNKNOWN = 64	创建文档参数错误	INVALID_ARGUMENT		代码地址
CODE_API_CREATE_FILE_GEN_FILE_GUID = 67	生成文件 GUID 失败	INTERNAL		代码地址
CODE_API_CREATE_FILE_CREATE_HISTORY_ERR = 71	非应用表格，创建 history err	INTERNAL		代码地址
CODE_MENTION_AT_LIST_CONTENT_EMPTY = 23	获取石墨文件内容中所有的 at 人列表接口中获取文件内容为空	INTERNAL		代码地址

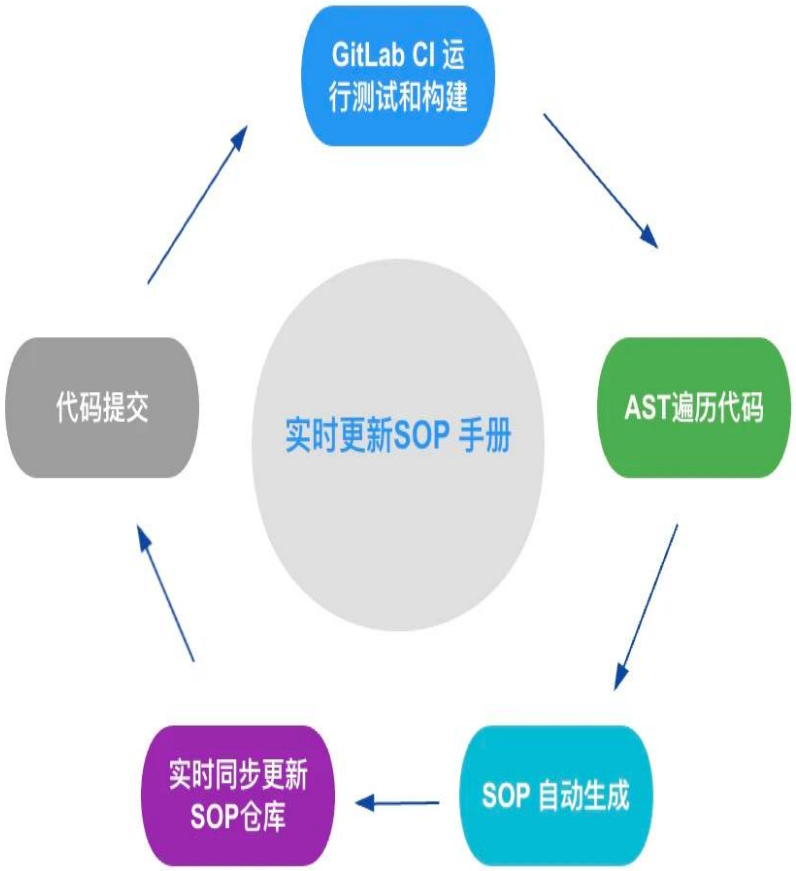
为什么恢复慢？-- 错误手册

经验

识别

自动化

- 自己写文档，代码和文档不一致
- 有错误码找不到代码对应地方
- 有错误码不知道是哪个版本的代码
- 错误码在代码里还有没有使用，错误码越变越多
- 错误码腐化，没用的错误码应该随版本下线
- 错误码的 SOP，国际化问题



相关链接

各种组件实现调试和高亮信息
https://github.com/gotomicro/ego
https://github.com/ego-component
快速定位错误
https://mp.weixin.qq.com/s/xXGjlSYOmY_uz6ZsbLSIEw
日志系统
https://github.com/clickvisual/clickvisual

感谢大家的收看