熔断中间件开发

联系QQ：2816010068，加入会员群

二叉树出品

目录

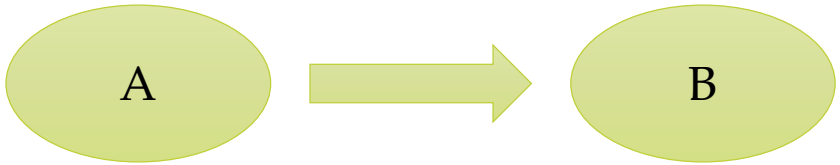
- 背景和问题
- 解决方案
- 熔断机制和原理详解
- Rpc熔断中间件开发

微服务架构的复杂性

- 请求失败原因：
  - **网络原因：**
    - 网络连接建立慢或失败
    - 网络请求超时
    - 服务过载
    - 网络抖动
  - **策略：重试**
  - **服务过载：**
    - **部分机器挂掉**
    - **流量突增，资源不足**
    - **部分网络挂掉**
  - **策略：~~重试解决不了~~**

微服务架构的复杂性

- 重试策略的问题：
  - 并发请求堵塞，关键资源直到超时才释放
    - 内存消耗
    - 线程消耗
    - 数据库连接被占用
  - 现象：
    - 内存耗尽
    - 线程被占光
    - 数据库连接被占光，请求被hang住
  - 后果：
    - 雪崩



解决方案

- 怎么解决？
  - 尽早拒绝
- 及时拒绝
  - 熔断机制

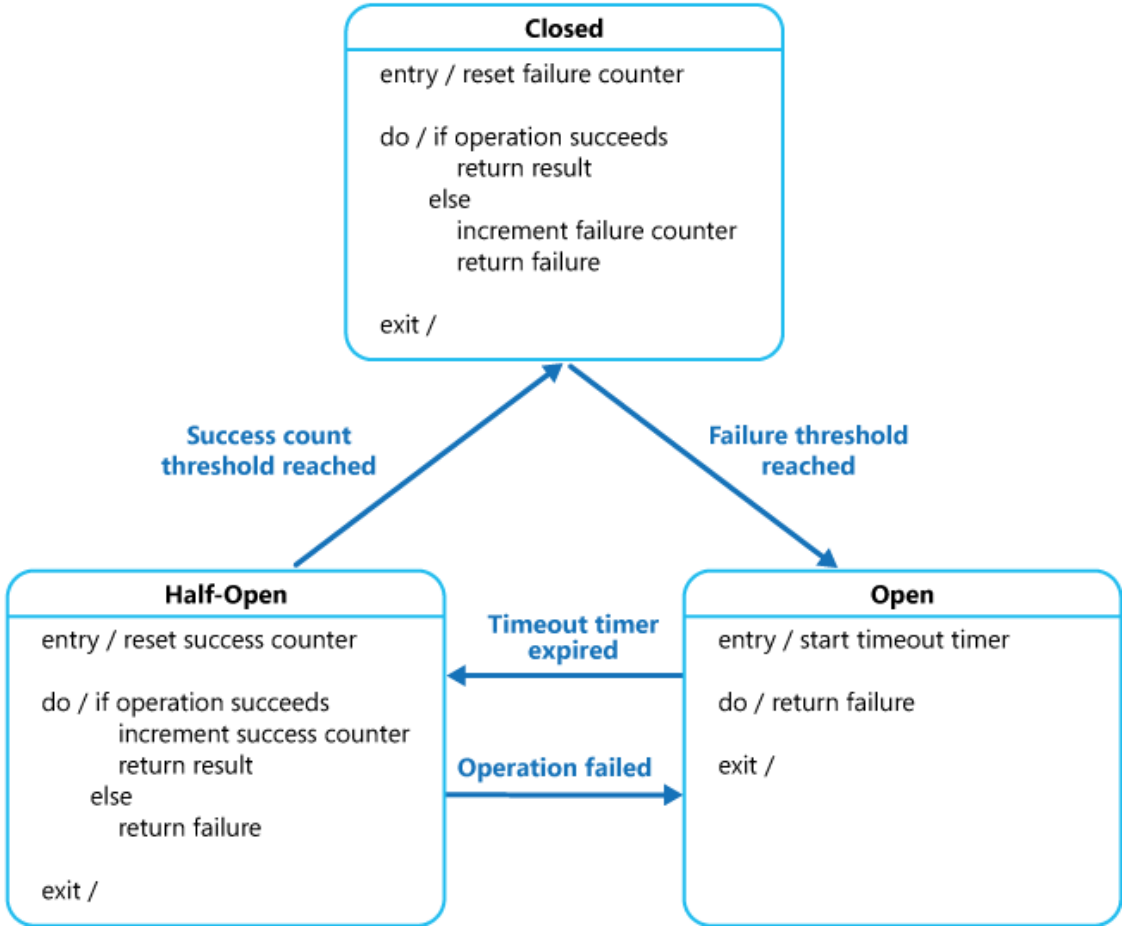
熔断机制

- 核心原理：
  - 阻止有潜在失败可能性的请求：
    - 如果一个请求，有比较大的失败可能，那么就应该及时拒绝这个请求
- 核心思路：
  - 对每一个发送请求的成功率进行预测
- 最佳方案：
  - 采用机器学习的方式进行预测
  - 机器学习本质上就是统计学，统计学玩的就是大数据
- **实现思路1**：
  - 针对每一个请求的结果，比如失败或成功进行统计
  - 在一定时间窗口内，如果失败率超过了一个比率，那么熔断器打开
  - 过一段时间后，熔断器再打开

实现思路1的问题

- 服务永远不会恢复
  - 比如：A->B，B由于某种原因5个实例全部挂掉
  - 熔断器检测到失败率过高，熔断器打开
  - 这时候，所有请求会被拒绝
  - 过了一段时间，B的一个实例恢复了
  - 熔断器打开，由于B只有一个实例，承载不了A的全部流量，瞬间被打挂
  - …
- 改进：
  - 引入半打开状态，Half-Open
  - 在Half-Open状态下，只有非常有限的请求会正常进行，这些请求任何一个失败，都会再次进入Open状态；这些请求如果全部成功，熔断器将会关闭

熔断器的状态机



Hystrix

- Netflix实现的容错库
- 功能强大
  - 过载保护：防止雪崩
  - 熔断器：快速失败，快速恢复
  - 并发控制：防止单个依赖把线程全部耗光
  - 超时控制：防止永远堵塞

Hystrix配置

- 配置
  - Timeout：超时配置，默认1000ms
  - MaxConcurrentRequests：并发控制，默认是10
  - SleepWindow：熔断器打开之后，冷却的时间，默认是500ms
  - RequestVolumeThreshold：一个统计窗口的请求数量，默认是20
  - ErrorPercentThreshold：失败百分比，默认是50%
- 触发条件：
  - 一个统计窗口内，请求数量大于RequestVolumeThreshold，且失败率大于ErrorPercentThreshold，才会触发熔断

Hystrix示例

```go
func main() {
    hystrix.ConfigureCommand("koala_rpc", hystrix.CommandConfig{
        Timeout:                10,
        MaxConcurrentRequests: 100,
        ErrorPercentThreshold: 25,
    })

    hystrix.Do("get_baidu", func() error {
        // talk to other services
        _, err := http.Get("https://www.baidu.com/")
        if err != nil {
            fmt.Println("get error")
            return err
        }
        return nil
    }, func(err error) error {
        fmt.Println("get an error, handle it, err:", err)
        return nil
    })

    time.Sleep(2 * time.Second) // 调用Go方法就是起了一个goroutine，这里要sleep
}
```

熔断中间件开发

- 基于Hystrix-go进行开发