【GO-Micro】micro 重试机制

作者：Allenxuxu
原文链接：https://ld246.com/article/1561018235398
来源网站：链滴

github完整代码地址 https://github.com/Allenxuxu/microservices

在分布式系统中，经常会有服务出现故障，所以良好的重试机制可以大大的提高系统的可用性。本文要分析micro的客户端重试机制，以及实例演示。

micro 重试实现

micro框架提供方法设置客户端重试的次数。

```
Client.Init(
    client.Retries(3),
)
```

当client请求失败时，客户端会根据selector的策略选择下一个节点重试请求。这样当一个服务实例故时，客户端可以自动调用另一个实例。

我们来看看micro 客户端内部重试的实现：

go-micro\client\rpc_client.go

```
func (r *rpcClient) Call(ctx context.Context, request Request, response interface{}, opts ...CallO
tion) error {
...
    //客户端call 调用函数， 在下面的循环中调用
    call := func(i int) error {
        // call backoff first. Someone may want an initial start delay
        t, err := callOpts.Backoff(ctx, request, i)
        if err != nil {
            return errors.InternalServerError("go.micro.client", "backoff error: %v", err.Error())
        }

        // only sleep if greater than 0
        if t.Seconds() > 0 {
            time.Sleep(t)
        }

        // 根据selector策略 选出 下一个节点
        node, err := next()
        if err != nil && err == selector.ErrNotFound {
            return errors.NotFound("go.micro.client", "service %s: %v", request.Service(), err.Error()

        } else if err != nil {
            return errors.InternalServerError("go.micro.client", "error getting next %s node: %v", r
quest.Service(), err.Error())
        }

        // 客户端调用
        err = rcall(ctx, node, request, response, callOpts)
        r.opts.Selector.Mark(request.Service(), node, err)
        return err
    }

    ch := make(chan error, callOpts.Retries+1)
```

```
    var gerr error
    //根据设定的**Retries**（重试次数）循环调用 call，如果执行成功，调用超时或者设置的**Retry*
函数执行出错则直接退出，不继续重试
    for i := 0; i <= callOpts.Retries; i++ {
        go func(i int) {
            ch <- call(i)
        }(i)

        select {
        case <-ctx.Done(): //超时
            return errors.Timeout("go.micro.client", fmt.Sprintf("call timeout: %v", ctx.Err()))
        case err := <-ch:
            // if the call succeeded lets bail early
            if err == nil {  //调用成功
                return nil
            }

            retry, rerr := callOpts.Retry(ctx, request, i, err)
            if rerr != nil {
                return rerr
            }

            if !retry {
                return err
            }

            gerr = err
        }
    }

    return gerr
}
```

micro将选举下一个节点，RPC调用封装到一个匿名函数中，然后根据设定的重试次数循环调用。如调用成功或者超时则直接返回，不继续重试。其中，当**callOpts**里设定的**Retry**函数执行失败，即第个返回值为false，或者第二个返回值为err不会nil时，也会退出循环直接返回。

我们来看下Retry是什么：

```
type CallOptions struct {
    Retry RetryFunc
}
```

client的CallOptions中定义了Retry，我们跳转到RetryFunc

go-micro\client\retry.go

```
// note that returning either false or a non-nil error will result in the call not being retried
type RetryFunc func(ctx context.Context, req Request, retryCount int, err error) (bool, error)

// RetryAlways always retry on error
func RetryAlways(ctx context.Context, req Request, retryCount int, err error) (bool, error) {
    return true, nil
}
```

```
// RetryOnError retries a request on a 500 or timeout error
func RetryOnError(ctx context.Context, req Request, retryCount int, err error) (bool, error) {
    if err == nil {
        return false, nil
    }

    e := errors.Parse(err.Error())
    if e == nil {
        return false, nil
    }

    switch e.Code {
    // retry on timeout or internal server error
    case 408, 500:
        return true, nil
    default:
        return false, nil
    }
}
```

从中我们可以发现，作者预实现了两个**Retry**函数：**RetryAlways**、**RetryOnError**。

RetryAlways直接返回**true, nil**，即不退出重试。

RetryOnError只有当e.Code（上一次RPC调用结果）为408或者500时才会返回**true, nil**，继续重试。

micro的默认**Retry**为**RetryOnError**，但是我们可以自定义并设置，下面的实验中将会演示。

```
    DefaultRetry = RetryOnError
    // DefaultRetries is the default number of times a request is tried
    DefaultRetries = 1
    // DefaultRequestTimeout is the default request timeout
    DefaultRequestTimeout = time.Second * 5
```

实验

当客户端请求另一个服务时，如果被请求的服务突然挂了，而此时客户端依旧会去请求，重试时客户会请求另一个实例（有一定几率还会请求同一个实例，因为默认的负载均衡策略是哈希随机）。

我们修改**api/user**下的服务，在**main**函数中设置客户端重试。

```
sClient := hystrixplugin.NewClientWrapper()(service.Options().Service.Client())
    sClient.Init(
        client.WrapCall(ocplugin.NewCallWrapper(t)),
        client.Retries(3),
        client.Retry(func(ctx context.Context, req client.Request, retryCount int, err error) (bool, e
ror) {
            log.Log(req.Method(), retryCount, " client retry")
            return true, nil
        }),
    )
```

然后，我们依次启动 micro网关，user API服务，hello SRV服务（启动两个实例）。

```
cd micro && make run
cd api/user && make run
```

```
cd srv/hello && make run
cd srv/hello && make run
```

我们通过kill -9 杀死其中一个hello服务，然后通过postman请求 **GET 172.0.0.1:8080/user/test**。

```
[GIN] 2019/05/14 - 14:52:20 | 200 |    1.253576ms |       127.0.0.1 | GET      /user/test
2019/05/14 14:52:48 Received Say.Anything API request
2019/05/14 14:52:48 0x19a1680 0 retry func
2019/05/14 14:52:48 msg:"Hello xuxu"
[GIN] 2019/05/14 - 14:52:48 | 200 |   13.821193ms |       127.0.0.1 | GET      /user/test
```

通过usr API服务的输出，我们可以看到重试一次后，客户端成功请求了另一个实例。

github完整代码地址 https://github.com/Allenxuxu/microservices