Gokit

Gophercon Korea 2015

anarcher

Gokit 소개

- Gokit에 관심을 가지게 된 이유
- Gokit의 구조

RPC Server 작성하기

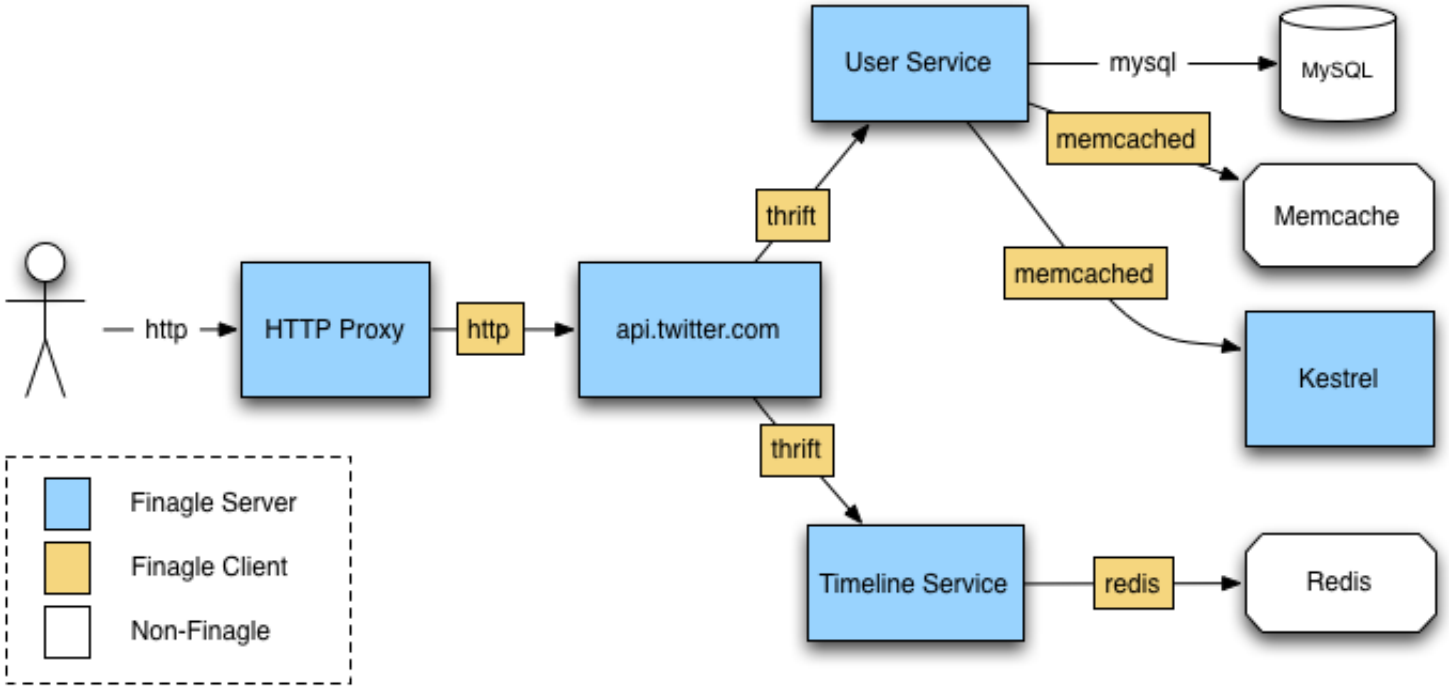
```
func main() {
    var add = new(Handler)
    s := rpc.NewServer()
    s.RegisterName("addsvc", add)
    s.HandleHTTP(rpc.DefaultRPCPath, rpc.DefaultDebugPath)
    go http.ListenAndServe(addr, s)
    time.Sleep(1 * time.Second)
    ret := ClientCall(1, 2)
    log.Println("ret:", ret)
}
```

Run

```
type AddRequest struct {
    A, B int64
}
type AddResponse struct {
    V int64
}
type Handler struct{}

func (h Handler) Add(req AddRequest, res *AddResponse) error {
    res.V = req.A + req.B
    return nil
}
```

- 입력받은 a,b를 더하는 비지니스 로직에 대한 RPC(net/rpc) 서버를 만들었다.
- 여기서, 실제로 서비스를 하기 위해서 무엇을 더해야 할까?



https://blog.twitter.com/2011/finagle-a-protocol-agnostic-rpc-system

(공통적인?!) 필요한 기능

여러 서버가 각기 다른 Transport(HTTP/JSON,gRPC,Thirft,Protobuf)을 사용하더라도, 비슷한 기능이 필요하다

- 운영중인 서버의 상태를 모니터링하고 싶다.

```
- package log
- package metric
- package tracing
```

- 다른 서버/서비스에 접속하기 위해,접속 가능한 접속 정보를 알거나 알려주어야 한다.

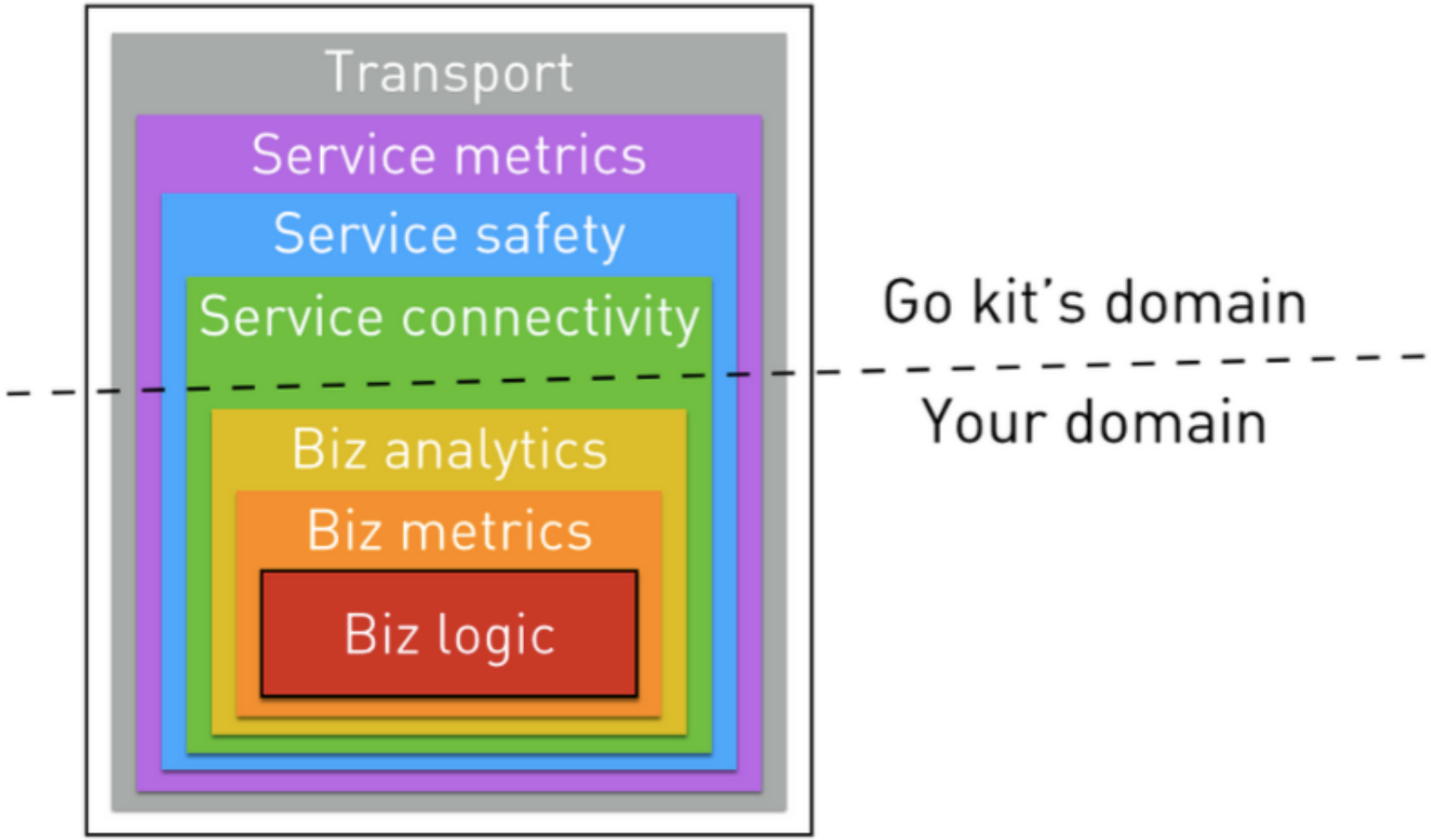
```
- service discovery
- package loadbalancer
```

- 네트워크 요청에 대한 제한을 주는 등으로 가능한 안정성을 부여하고 싶다.

```
- package ratelimit
- package circuitbreaker
```

- 기타 등등

Go kit : a distributed programming toolkit



http://www.gophercon.com/talks/go-kit/

package endpoint

```
// Endpoint is the fundamental building block of servers and clients.
// It represents a single RPC method.
type Endpoint func(ctx context.Context, request interface{}) (response interface{}, err error)

// Middleware is a chainable behavior modifier for endpoints.
type Middleware func(Endpoint) Endpoint
```

- Endpoint은 하나의 RPC 메소드를 나타낸다
- Gokit의 가장 기본적인 인터페이스
- Middleware : Endpoint을 받아서 Endpoint을 반환하는 함수로 Endpoint의 기능을 확장한다.

```
- ratelimit
- tracing
- circuitbreaker
- loadbalancer
- ...
```

StringService

```
type StringService interface {
    Uppercase(string) (string, error)
    Count(string) int
}

type stringService struct{}

func (stringService) Uppercase(s string) (string, error) {
    if s == "" {
        return "", ErrEmpty
    }
    return strings.ToUpper(s), nil
}

func (stringService) Count(s string) int {
    return len(s)
}
```

- StringService 인터페이스로 비지니스 모델을 정의한다.

Request와 Response

```
type UppercaseRequest struct {
    S string `json:"s"`
}

type UppercaseResponse struct {
    V   string `json:"v"`
    Err error  `json:"err"`
}

type CountRequest struct {
    S string `json:"s"`
}

type CountResponse struct {
    V int `json:"v"`
}
```

- `StringService` 의 RPC 메소드의 요청(`Request`)와 응답(`Response`)을 `struct` 으로 구성

Endpoint

```
type Endpoint func(ctx context.Context, request interface{}) (response interface{}, err error)
```

```
func makeUppercaseEndpoint(svc StringService) endpoint.Endpoint {
    return func(ctx context.Context, request interface{}) (interface{}, error) {
        req := request.(UppercaseRequest)
        v, err := svc.Uppercase(req.S)
        return UppercaseResponse{v, err}, nil
    }
}

func makeCountEndpoint(svc StringService) endpoint.Endpoint {
    return func(ctx context.Context, request interface{}) (interface{}, error) {
        req := request.(CountRequest)
        v := svc.Count(req.S)
        return CountResponse{v}, nil
    }
}
```

Transport

```
// NetRpc's handler
type NetRpcBinding struct {
    Context           context.Context
    uppercaseEndpoint endpoint.Endpoint
    countEndpoint     endpoint.Endpoint
}
```

- Endpoint 을 RPC 라이브러리(net/rpc,grpc,thrift,...)에 필요한 형태로 사용할수 있도록
- HTTP 의 경우, `github.com/go-kit/kit/transport/http` 에 helper struct가 있다

```
uppercaseHandler := httptransport.Server{
    Context:    ctx,
    Endpoint:   makeUppercaseEndpoint(svc),
    DecodeFunc: decodeUppercaseRequest,
    EncodeFunc: encodeResponse,
}
countHandler := httptransport.Server{
    Context:    ctx,
    Endpoint:   makeCountEndpoint(svc),
    DecodeFunc: decodeCountRequest,
    EncodeFunc: encodeResponse,
}
```

context.Context

- http://blog.golang.org/context

```
type Context interface {
    Done() <-chan struct{}                    // closed when this Context is canceled
    Err() error                               // why this Context was canceled
    Deadline() (deadline time.Time, ok bool) // when this Context will be canceled
    Value(key interface{}) interface{}        // data associated with this Context
}
```

- Context들은 계층구조를 가진다 (상위 Context의 value가 파생된 Context에게 전달된다)

```
type CancelFunc
type Context
    func Background() Context
    func TODO() Context
    func WithCancel(parent Context) (ctx Context, cancel CancelFunc)
    func WithDeadline(parent Context, deadline time.Time) (Context, CancelFunc)
    func WithTimeout(parent Context, timeout time.Duration) (Context, CancelFunc)
    func WithValue(parent Context, key interface{}, val interface{}) Context
```

```
func (n NetRpcBinding) Uppercase(req UppercaseRequest, res *UppercaseResponse) error {
    ctx, cancel := context.WithCancel(n.Context)
    defer cancel()
    responses := make(chan UppercaseResponse, 1)
    errs := make(chan error, 1)

    go func() {
        resp, err := n.uppercaseEndpoint(ctx, req)
        if err != nil {
            errs <- err
            return
        }
        responses <- resp.(UppercaseResponse)
    }()

    select {
    case <-ctx.Done():
        return context.DeadlineExceeded
    case err := <-errs:
        return err
    case resp := <-responses:
        (*res) = resp
        return nil
    }
}
```

```
func (n NetRpcBinding) Count(req CountRequest, res *CountResponse) error {
    ctx, cancel := context.WithCancel(n.Context)
    defer cancel()
    responses := make(chan CountResponse, 1)
    errs := make(chan error, 1)

    go func() {
        resp, err := n.countEndpoint(ctx, req)
        if err != nil {
            errs <- err
            return
        }
        responses <- resp.(CountResponse)
    }()

    select {
    case <-ctx.Done():
        return context.DeadlineExceeded
    case err := <-errs:
        return err
    case resp := <-responses:
        (*res) = resp
        return nil
    }
}
```

Main

```
func main() {
    ctx := context.Background()
    svc := stringService{}
    netRpcBinding := NetRpcBinding{ctx, makeUppercaseEndpoint(svc), makeCountEndpoint(svc)}

    s := rpc.NewServer()
    s.RegisterName("stringsvc", netRpcBinding)
    s.HandleHTTP(rpc.DefaultRPCPath, rpc.DefaultDebugPath)
    go func() {
        err := http.ListenAndServe(":8080", s)
        if err != nil {
            log.Fatal(err)
        }
    }()
    time.Sleep(1 * time.Second)
    client, _ := rpc.DialHTTP("tcp", "localhost:8080") // sorry for ignore the error
    clientEndpoint := NewNetRpcClient(client)
    req := UppercaseRequest{S: "gokit!"}
    res, err := clientEndpoint(ctx, req)
    log.Println("res:", res.(UppercaseResponse).V, "err:", err)
}
```

Run

Client

```
func NewNetRpcClient(c *rpc.Client) endpoint.Endpoint {
    return func(ctx context.Context, request interface{}) (interface{}, error) {
        errs := make(chan error, 1)
        responses := make(chan interface{}, 1)
        go func() {
            var response UppercaseResponse
            if err := c.Call("stringsvc.Uppercase", request, &response); err != nil {
                errs <- err
                return
            }
            responses <- response
        }()
        select {
        case <-ctx.Done():
            return nil, context.DeadlineExceeded
        case err := <-errs:
            return nil, err
        case resp := <-responses:
            return resp, nil
        }
    }
}
```

- Client도 역시...

Example: LRU Cache Endpoint

```
type CacheKeyFunc func(request interface{}) (interface{}, bool)
func NewLRUCache(cache *lru.Cache, cacheKey CacheKeyFunc) endpoint.Middleware {
    return func(next endpoint.Endpoint) endpoint.Endpoint {
        return func(ctx context.Context, request interface{}) (interface{}, error) {
            key, ok := cacheKey(request)
            if !ok {
                return next(ctx, request)
            }
            val, ok := cache.Get(key)
            if ok {
                fmt.Println("Return from cache", request, val)
                return val, nil
            }
            val, err := next(ctx, request)
            if err == nil {
                cache.Add(key, val)
            }
            fmt.Println("Return from endpoint", request, val)
            return val, err
        }
    }
}
```

Run

```
    cacheKeyFunc := func(request interface{}) (interface{}, bool) {
        if req, ok := request.(UppercaseRequest); ok {
            return req.S, true
        }
        return nil, false
    }

    cache, _ := lru.New(10)
    e := makeUppercaseEndpoint(svc)
    e = NewLRUCache(cache, cacheKeyFunc)(e)

    req := UppercaseRequest{"gophercon!"}
    resp, err := e(context.Background(), req)
    fmt.Println("resp", resp.(UppercaseResponse).V, "err", err)

    resp, err = e(context.Background(), req)
    fmt.Println("resp", resp.(UppercaseResponse).V, "err", err)
```

package logging

```
    var logger log.Logger
    logger = log.NewLogfmtLogger(os.Stderr)
    logger = log.NewContext(logger).With("ts", log.DefaultTimestampUTC)

    uppercase := makeUppercaseEndpoint(svc)
    count := makeCountEndpoint(svc)
    uppercase = logging(log.NewContext(logger).With("method", "uppercase"))(uppercase)
    count = logging(log.NewContext(logger).With("method", "count"))(count)

    //net/rpc
    netRpcBinding := NetRpcBinding{ctx, uppercase, count}
```

Run

```
func logging(logger log.Logger) endpoint.Middleware {
    return func(next endpoint.Endpoint) endpoint.Endpoint {
        return func(ctx context.Context, request interface{}) (interface{}, error) {
            logger.Log("msg", "calling endpoint")
            defer logger.Log("msg", "called endpoint")
            return next(ctx, request)
        }
    }
}
```

package metric

- counters , gauges , histograms 에 대한 공통적인 인터페이스를 제공
- expvar , statsd , prometheus 에 대한 어댑터 제공

```
requests := metrics.NewMultiCounter(
        expvar.NewCounter("requests"),
        statsd.NewCounter(ioutil.Discard, "requests_total", time.Second),
        prometheus.NewCounter(stdprometheus.CounterOpts{
                Namespace: "addsvc",
                Subsystem: "add",
                Name:      "requests_total",
                Help:      "Total number of received requests.",
                }, []string{}),
)

func metric(requests metrics.Counter) endpoint.Middleware {
    return func(next endpoint.Endpoint) endpoint.Endpoint {
            return func(ctx context.Context, request interface{}) (interface{}, error) {
                requests.Add(1)
                return next(ctx,request)
            }
    }
}
```

package ratelimit

```
// NewTokenBucketLimiter returns an endpoint.Middleware that acts as a rate
// limiter based on a token-bucket algorithm. Requests that would exceed the
// maximum request rate are simply rejected with an error.

func NewTokenBucketLimiter(tb *ratelimit.Bucket) endpoint.Middleware {
    return func(next endpoint.Endpoint) endpoint.Endpoint {
        return func(ctx context.Context, request interface{}) (interface{}, error) {
            if tb.TakeAvailable(1) == 0 {
                return nil, ErrLimited
            }
            return next(ctx, request)
        }
    }
}
```

- github.com/juju/ratelimit

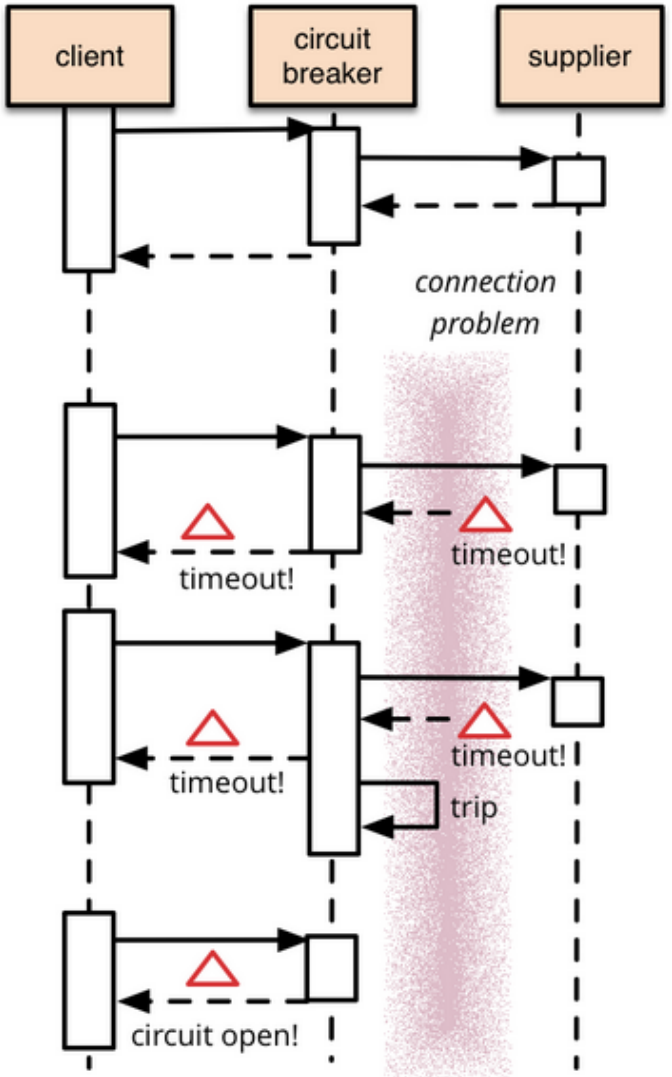
```
    e = makeUppercaseEndpoint(svc)
    e = ratelimit.NewTokenBucketThrottler(jujuratelimit.NewBucketWithRate(1, 1), s)(e)
```

package circuitbreaker

```
func HandyBreaker(cb breaker.Breaker) endpoint.Middleware {
    return func(next endpoint.Endpoint) endpoint.Endpoint {
        return func(ctx context.Context, request interface{}) (response interface{}, err error)
{
            if !cb.Allow() {
                return nil, breaker.ErrCircuitOpen
            }

            defer func(begin time.Time) {
                if err == nil {
                    cb.Success(time.Since(begin))
                } else {
                    cb.Failure(time.Since(begin))
                }
            }(time.Now())

            response, err = next(ctx, request)
            return
        }
    }
}
```



http://martinfowler.com/bliki/CircuitBreaker.html

package loadbalancer

```
import (
    "github.com/go-kit/kit/loadbalancer"
    "github.com/go-kit/kit/loadbalancer/dnssrv"
)

func main() {
    // Construct a load balancer for foosvc, which gets foosvc instances by
    // polling a specific DNS SRV name.
    p := dnssrv.NewPublisher("foosvc.internal.domain", 5*time.Second, fooFactory, logger)
    lb := loadbalancer.NewRoundRobin(p)

    // Get a new endpoint from the load balancer.
    endpoint, err := lb.Endpoint()
    if err != nil {
        panic(err)
    }

    // Use the endpoint to make a request.
    response, err := endpoint(ctx, request)
}

func fooFactory(instance string) (endpoint.Endpoint, error) {
    // Convert an instance (host:port) to an endpoint, via a defined transport binding.
}
```

```
func main() {
    p := dnssrv.NewPublisher("foosvc.internal.domain", 5*time.Second, fooFactory, logger)
    lb := loadbalancer.NewRoundRobin(p)
    endpoint := loadbalancer.Retry(3, 5*time.Seconds, lb)

    response, err := endpoint(ctx, request) // requests will be automatically load balanced
}
```

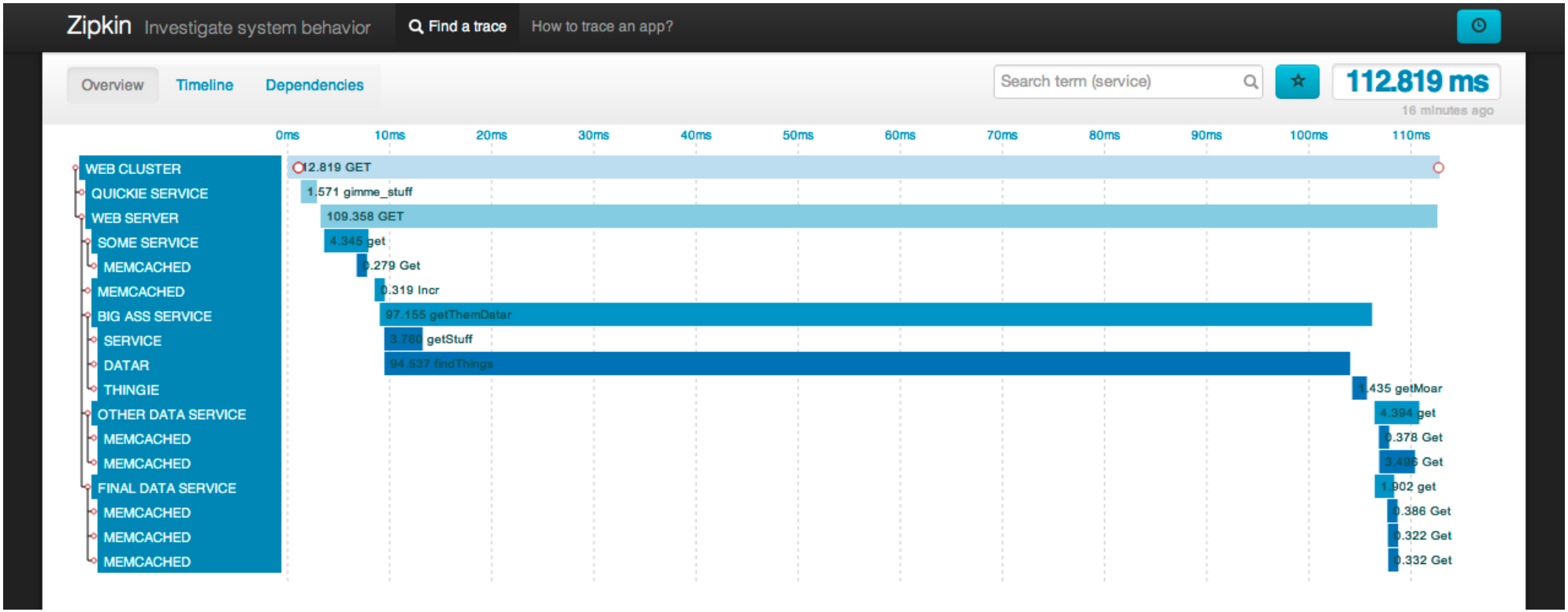
- 현재 dnssrv 와 static 지원
- consul , etcd , zookeeper 등 지원 예정

package tracing

```
func AnnotateServer(newSpan NewSpanFunc, c Collector) endpoint.Middleware {
    return func(next endpoint.Endpoint) endpoint.Endpoint {
        return func(ctx context.Context, request interface{}) (interface{}, error) {
            span, ok := fromContext(ctx)
            if !ok {
                span = newSpan(newID(), newID(), 0)
                ctx = context.WithValue(ctx, SpanContextKey, span)
            }
            span.Annotate(ServerReceive)
            defer func() { span.Annotate(ServerSend); c.Collect(span) }()
            return next(ctx, request)
        }
    }
}

// Server-side
var server endpoint.Endpoint
server = makeEndpoint() // for your service
server = zipkin.AnnotateServer(spanFunc, collector)(server)
go serveViaHTTP(server)
```

- Appdash 은 계획중



https://blog.twitter.com/2012/distributed-systems-tracing-with-zipkin

Thank you

anarcher
anarcher@gmail.com (mailto:anarcher@gmail.com)