Go培训第12天

tony

Outline

1. etcd介绍与使用

2. ElasticSearch介绍与使用

3. 课后作业

etcd介绍与使用

1. etcd介绍

概念：高可用的分布式key-value存储，可以用于配置共享和服务发现。

类似项目：zookeeper和consul

开发语言：Go

接口：提供restful的http接口，使用简单

实现算法：基于raft算法的强一致性、高可用的服务存储目录

etcd介绍与使用

2. etcd的应用场景

a. 服务发现和服务注册

b. 配置中心

c. 分布式锁

d. master选举

etcd介绍与使用

3. etcd搭建

a. 下载etcd release版本：https://github.com/coreos/etcd/releases/

b. ./bin/etcd即可以启动etcd

c. 使用etcdctl工具更改配置

etcd介绍与使用

4. context使用介绍

    a. 如何控制goroutine的超时?

    b. 如何保存上下文数据?

5. 使用context处理超时

```
ctx, cancel := context.WithTimeout(context.Background(), 2*time.Second)
```

```
package main

import (
	"context"
	"fmt"
	"io/ioutil"
	"net/http"
	"time"
)
type Result struct {
	r   *http.Response
	err error
}
func process() {
	ctx, cancel := context.WithTimeout(context.Background(), 2*time.Second)
	defer cancel()
	tr := &http.Transport{}
	client := &http.Client{Transport: tr}
	c := make(chan Result, 1)
	req, err := http.NewRequest("GET", "http://google.com", nil)
	if err != nil {
		fmt.Println("http request failed, err:", err)
		return
	}
	go func() {
		resp, err := client.Do(req)
		pack := Result{r: resp, err: err}
		c <- pack
	}()
	select {
	case <-ctx.Done():
		tr.CancelRequest(req)
		<-c
		fmt.Println("Timeout!")
	case res := <-c:
		defer res.r.Body.Close()
		out, _ := ioutil.ReadAll(res.r.Body)
		fmt.Printf("Server Response: %s", out)
	}
	return
}
func main() {
	process()
}
```

6. 使用context保存上下文

```
package main

import (
	"context"
	"fmt"
)

func add(ctx context.Context, a, b int) int {
	traceId := ctx.Value("trace_id").(string)
	fmt.Printf("trace_id:%v\n", traceId)
	return a + b
}

func calc(ctx context.Context, a, b int) int {
	traceId := ctx.Value("trace_id").(string)
	fmt.Printf("trace_id:%v\n", traceId)
	return add(ctx, a, b)
}

func main() {
	ctx := context.WithValue(context.Background(), "trace_id", "123456")
	calc(ctx, 388, 200)
}
```

etcd介绍与使用

7. etcd使用示例

```
package main

import (
	"fmt"
	"github.com/coreos/etcd/clientv3"
	"time"
)

func main() {

	cli, err := clientv3.New(clientv3.Config{
		Endpoints:   []string{"localhost:2379", "localhost:22379", "localhost:32379"},
		DialTimeout: 5 * time.Second,
	})
	if err != nil {
		fmt.Println("connect failed, err:", err)
		return
	}

	fmt.Println("connect succ")
	defer cli.Close()
}
```

etcd介绍与使用

```
package main

import (
	"context"
	"fmt"
	"github.com/coreos/etcd/clientv3"
	"time"
)

func main() {

	cli, err := clientv3.New(clientv3.Config{
		Endpoints:   []string{"localhost:2379", "localhost:22379", "localhost:32379"},
		DialTimeout: 5 * time.Second,
	})
	if err != nil {
		fmt.Println("connect failed, err:", err)
		return
	}

	fmt.Println("connect succ")
	defer cli.Close()

	ctx, cancel := context.WithTimeout(context.Background(), time.Second)
	_, err = cli.Put(ctx, "/logagent/conf/", "sample_value")
	cancel()
	if err != nil {
		fmt.Println("put failed, err:", err)
		return
	}

	ctx, cancel = context.WithTimeout(context.Background(), time.Second)
	resp, err := cli.Get(ctx, "/logagent/conf/")
	cancel()
	if err != nil {
		fmt.Println("get failed, err:", err)
		return
	}
	for _, ev := range resp.Kvs {
		fmt.Printf("%s : %s\n", ev.Key, ev.Value)
	}
}
```

etcd介绍与使用

```
package main

import (
	"context"
	"fmt"
	"github.com/coreos/etcd/clientv3"
	"time"
)

func main() {

	cli, err := clientv3.New(clientv3.Config{
		Endpoints:   []string{"localhost:2379", "localhost:22379", "localhost:32379"},
		DialTimeout: 5 * time.Second,
	})
	if err != nil {
		fmt.Println("connect failed, err:", err)
		return
	}

	fmt.Println("connect succ")
	defer cli.Close()

	rch := cli.Watch(context.Background(), "/logagent/conf/")
	for wresp := range rch {
		for _, ev := range wresp.Events {
			fmt.Printf("%s %q : %q\n", ev.Type, ev.Kv.Key, ev.Kv.Value)
		}
	}
}
```

kafka消费者示例代码

```
package main

import (
	"fmt"
	"github.com/Shopify/sarama"
	"strings"
	"sync"
	"time"
)
var (
	wg sync.WaitGroup
)
func main() {

	consumer, err := sarama.NewConsumer(strings.Split("192.168.31.177:9092", ","), nil)
	if err != nil {
		fmt.Println("Failed to start consumer: %s", err)
		return
	}
	partitionList, err := consumer.Partitions("nginx_log")
	if err != nil {
		fmt.Println("Failed to get the list of partitions: ", err)
		return
	}
	fmt.Println(partitionList)

	for partition := range partitionList {
		pc, err := consumer.ConsumePartition("nginx_log", int32(partition), sarama.OffsetNewest)
		if err != nil {
			fmt.Printf("Failed to start consumer for partition %d: %s\n", partition, err)
			return
		}
		defer pc.AsyncClose()
		go func(sarama.PartitionConsumer) {
			for msg := range pc.Messages() {
				fmt.Printf("Partition:%d, Offset:%d, Key:%s, Value:%s", msg.Partition, msg.Offset, string(msg.Key), string(msg.Value))
				fmt.Println()
			}
		}(pc)
	}
	time.Sleep(time.Hour)
	consumer.Close()
}
```

kafka消费代码优化

8. sync.WaitGroup介绍

1) 等待一组goroutine结束

2) 使用Add方法设置等待的数量加1

3) 使用Done方法设置等待的数量减1

4) 当等待的数量等于0时，Wait函数返回

kafka消费代码优化

8. sync.WaitGroup实例

```
package main

import (
	"fmt"
	"sync"
	"time"
)

func main() {
	wg := sync.WaitGroup{}
	wg.Add(10)

	for i := 0; i < 10; i++ {
		go calc(&wg, i)
	}

	wg.Wait()
	fmt.Println("all goroutine finish")
}
func calc(w *sync.WaitGroup, i int) {
	fmt.Println("calc:", i)
	time.Sleep(time.Second)
	w.Done()
}
```

ElasticSearch介绍与使用

9. ElasticSearch安装

1) 下载ES，下载地：github.com/elastic/elasticsearch

2) 修改config/elasticsearch.ymal配置：
network.host: 本地ip
node.name:node_1

3) 启动es， ./bin/elasticsearch.bat

```
package main

import (
	"fmt"
	elastic "gopkg.in/olivere/elastic.v2"
)

type Tweet struct {
	User    string
	Message string
}

func main() {
	client, err := elastic.NewClient(elastic.SetSniff(false), elastic.SetURL("http://192.168.31.177:9200/"))
	if err != nil {
		fmt.Println("connect es error", err)
		return
	}

	fmt.Println("conn es succ")

	tweet := Tweet{User: "olivere", Message: "Take Five"}
	_, err = client.Index().
		Index("twitter").
		Type("tweet").
		Id("1").
		BodyJson(tweet).
		Do()
	if err != nil {
		// Handle error
		panic(err)
		return
	}

	fmt.Println("insert succ")
}
```

获取服务器信息

10. 获取服务器信息

1) github.com/shirou/gopsutil

课后作业

1. 把今天的日志收集客户端，自己实现一遍