Go培训第11天

tony

Outline

1. 日志收集系统设计

2. 日志客户端开发

3. 课后作业

日志收集系统架构

1. 项目背景

   a. 每个系统都有日志，当系统出现问题时，需要通过日志解决问题

   b. 当系统机器比较少时，登陆到服务器上查看即可满足

   c. 当系统机器规模巨大，登陆到机器上查看几乎不现实

日志收集系统架构

2. 解决方案

a. 把机器上的日志实时收集，统一的存储到中心系统

b. 然后再对这些日志建立索引，通过搜索即可以找到对应日志

c. 通过提供界面友好的web界面，通过web即可以完成日志搜索

日志收集系统架构

3. 面临的问题

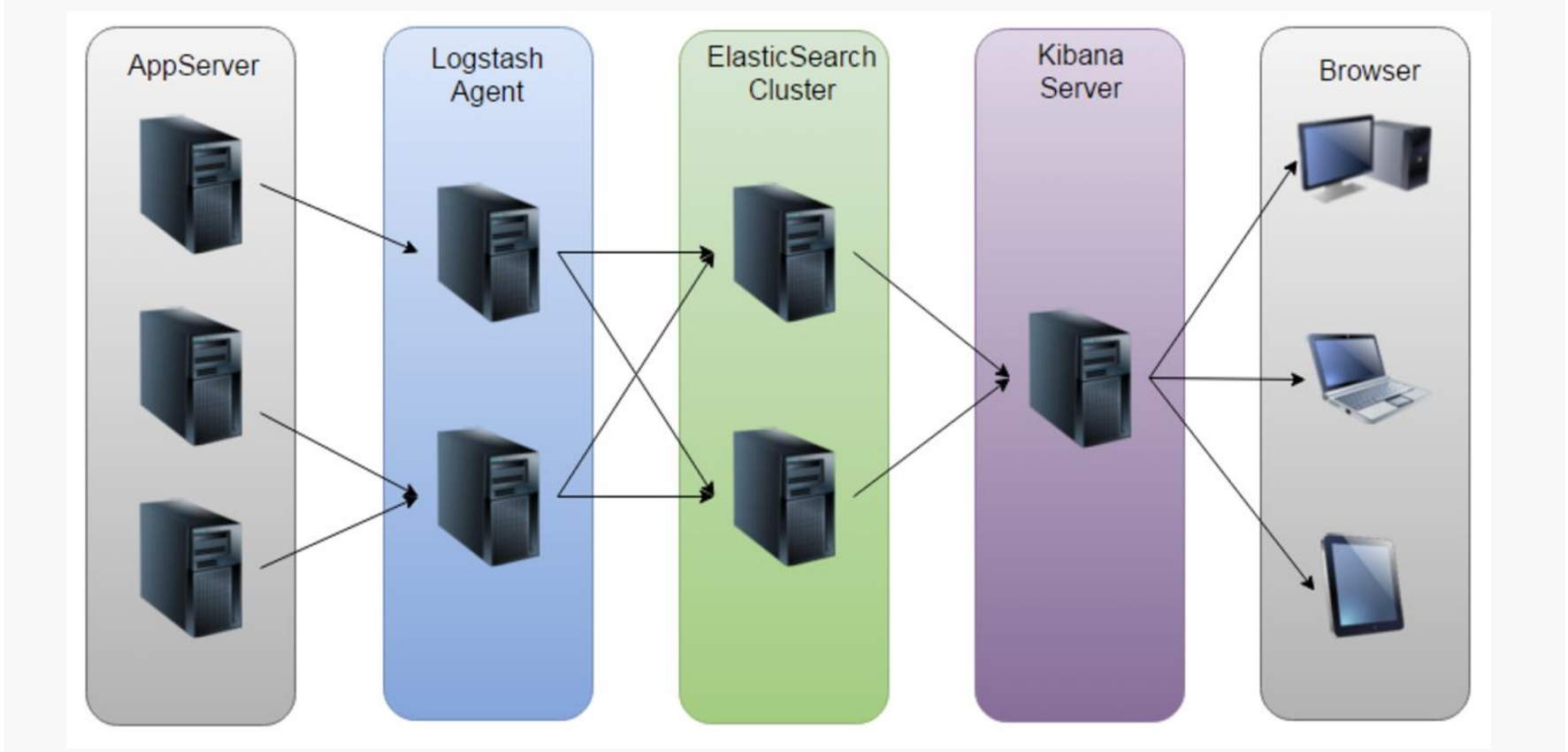
   a. 实时日志量非常大，每天几十亿条

   b. 日志准实时收集，延迟控制在分钟级别

   c. 能够水平可扩展

日志收集系统架构

4. 业界方案ELK



日志收集系统架构

5. elk方案问题

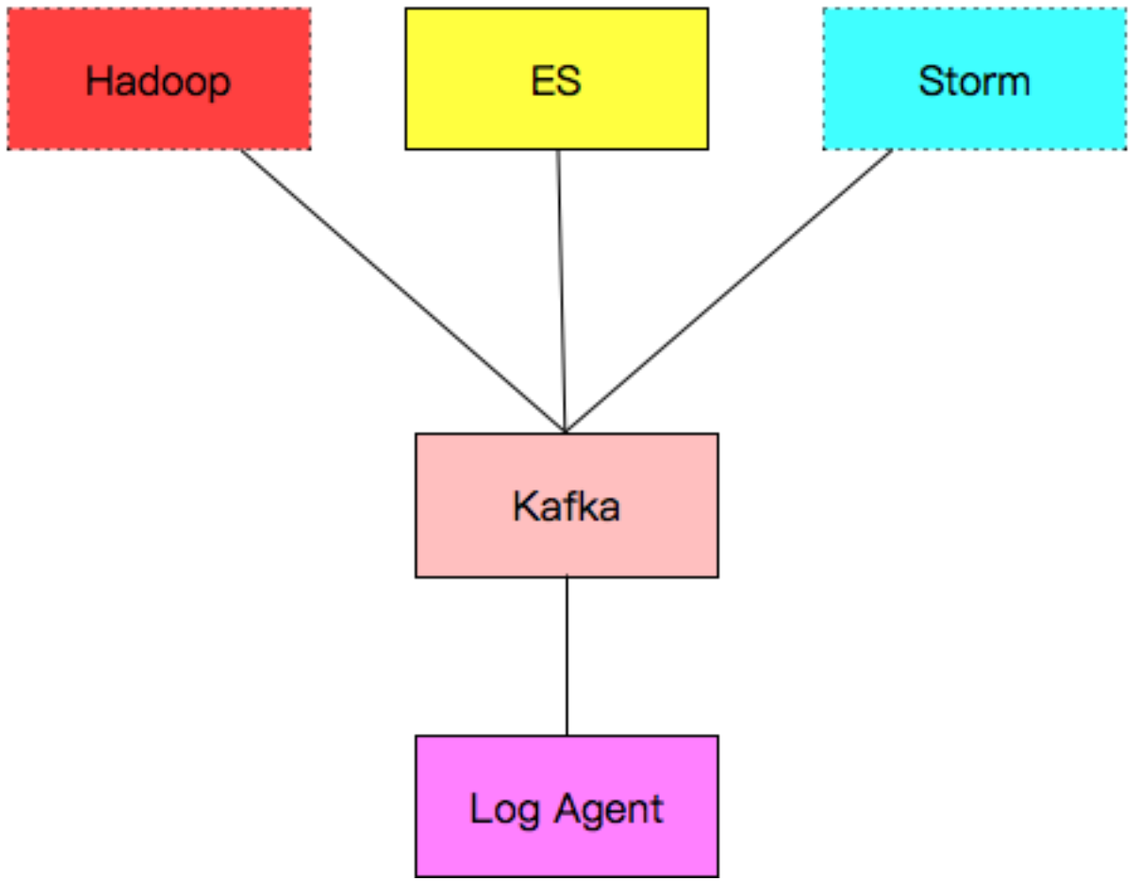
a. 运维成本高，每增加一个日志收集，都需要手动修改配置

b. 监控缺失，无法准确获取logstash的状态

c. 无法做定制化开发以及维护

日志收集系统架构

6. 日志收集系统设计



日志收集系统架构

7. 各组件介绍

a. Log Agent，日志收集客户端，用来收集服务器上的日志

b. Kafka，高吞吐量的分布式队列，linkin开发，apache顶级开源项目

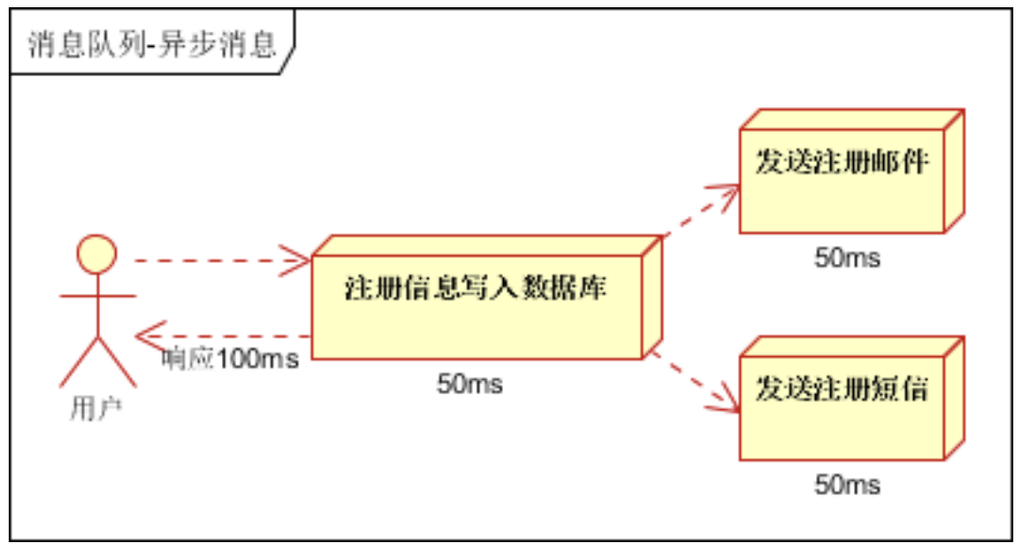
c. ES，elasticsearch，开源的搜索引擎，提供基于http restful的web接口

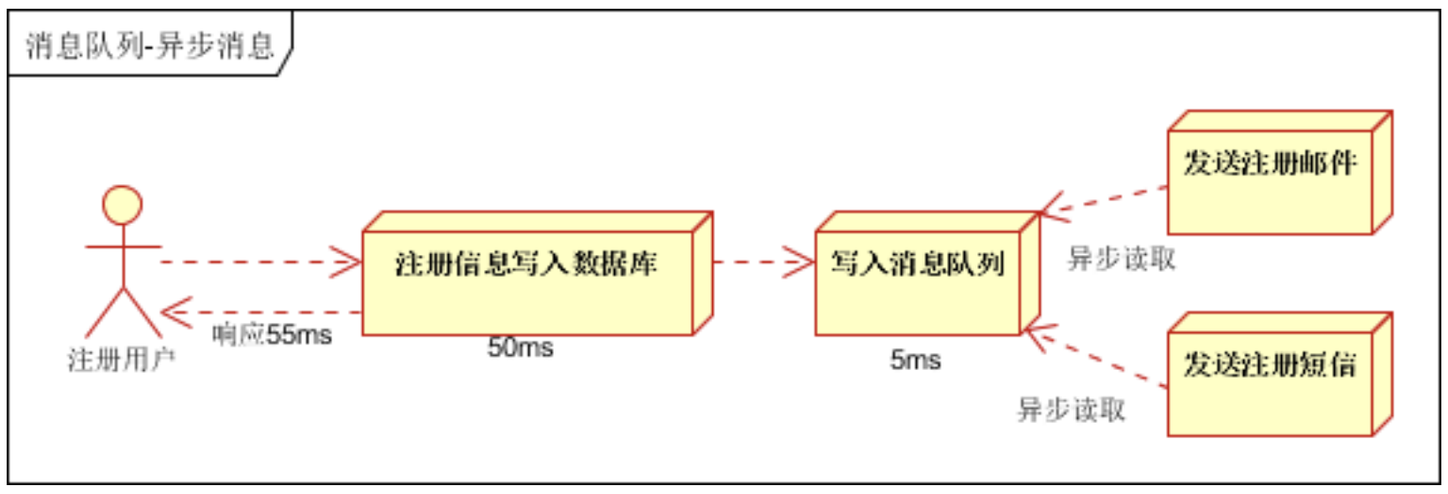
d. Hadoop，分布式计算框架，能够对大量数据进行分布式处理的平台

日志收集系统架构

7.1 kafka应用场景

1. 异步处理, 把非关键流程异步化，提高系统的响应时间和健壮性

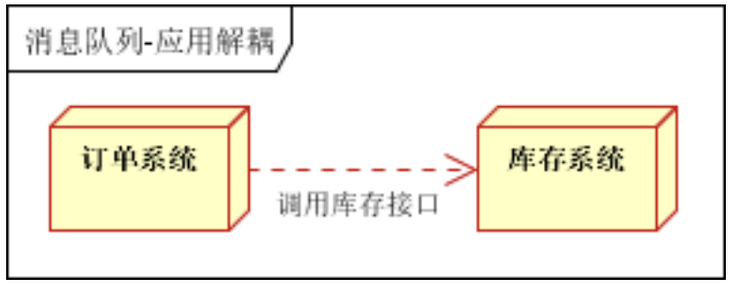


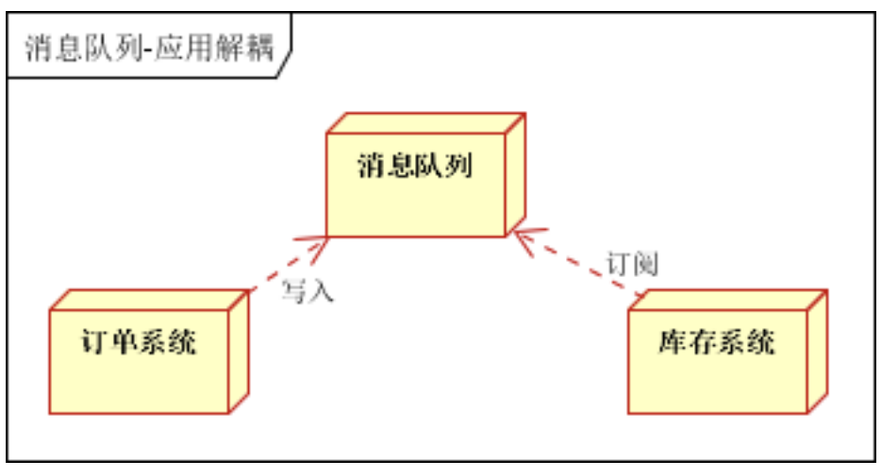


日志收集系统架构

7.1 kafka应用场景

2. 应用解耦,通过消息队列,

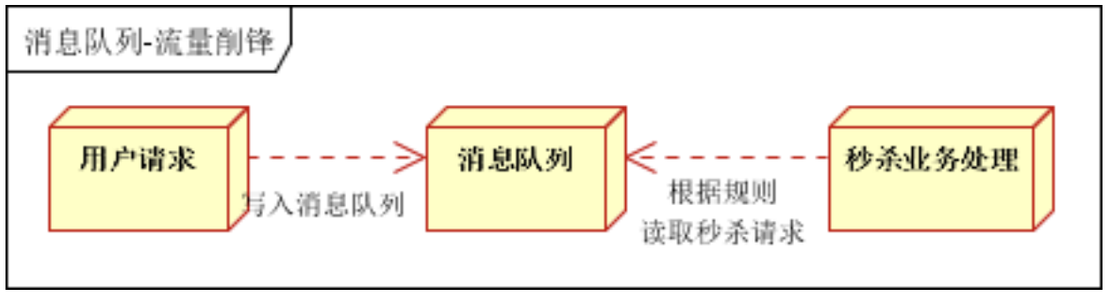




日志收集系统架构

7.1 kafka应用场景

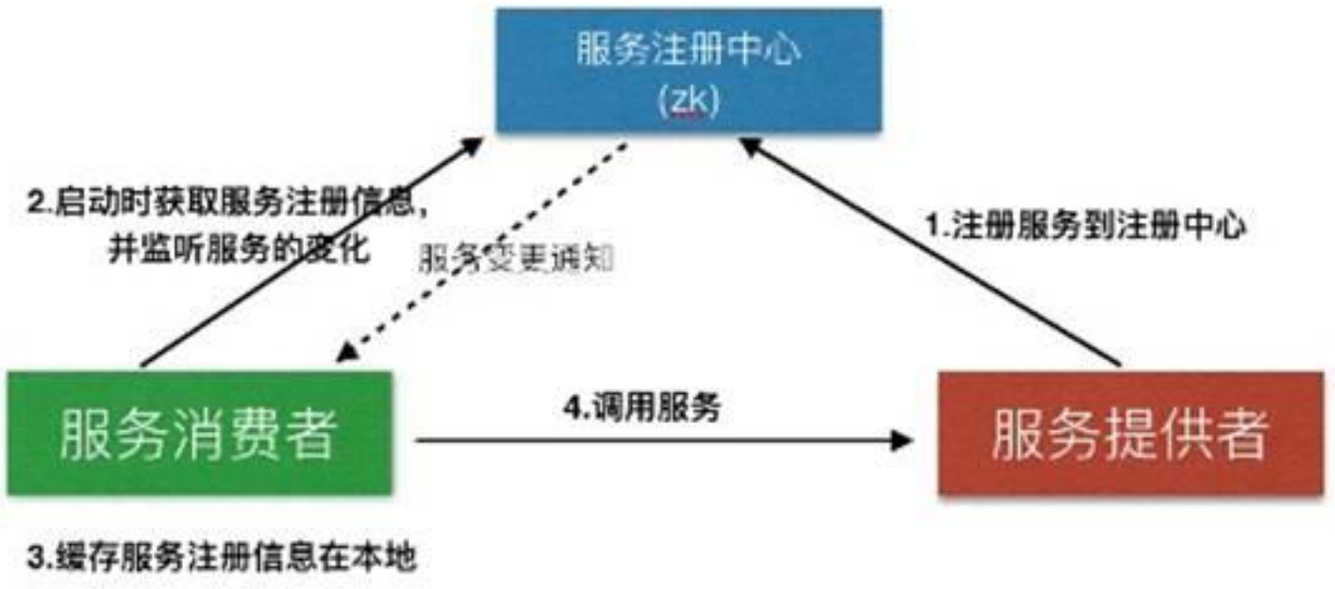
3. 流量削峰



日志收集系统架构

7.2 zookeeper应用场景

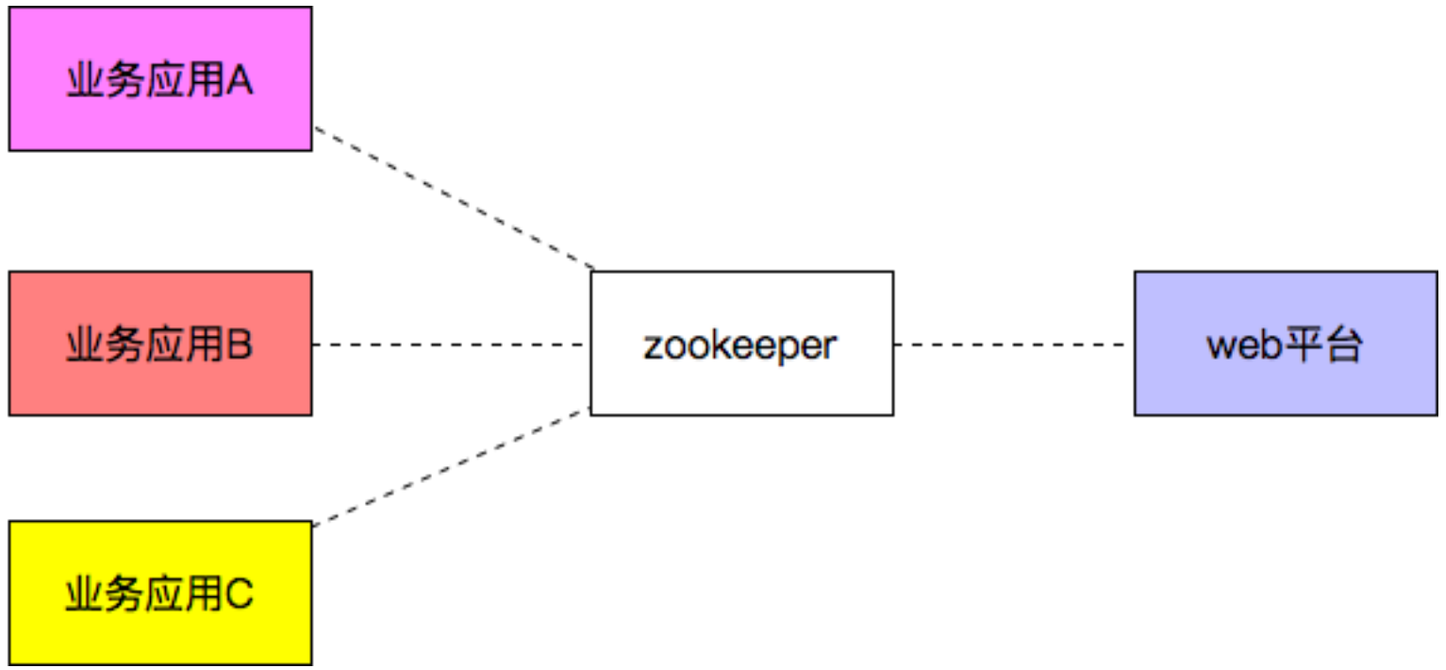
1. 服务注册&服务发现



日志收集系统架构

7.2 zookeeper应用场景

2. 配置中心



7.2 zookeeper应用场景

3. 分布式锁

- Zookeeper是强一致的
- 多个客户端同时在Zookeeper上创建相同znode，只有一个创建成功

日志收集系统架构

8. 安装kafka

a. 安装JDK，从oracle下载最新的SDK安装

b. 安装zookeeper3.3.6，下载地址：http://apache.fayea.com/zookeeper/

1）mv conf/zoo_sample.cfg conf/zoo.cfg

2）编辑 conf/zoo.cfg，修改dataDir=D:\zookeeper-3.3.6\data\

3）运行bin/zkServer.cmd

c. 安装kafka

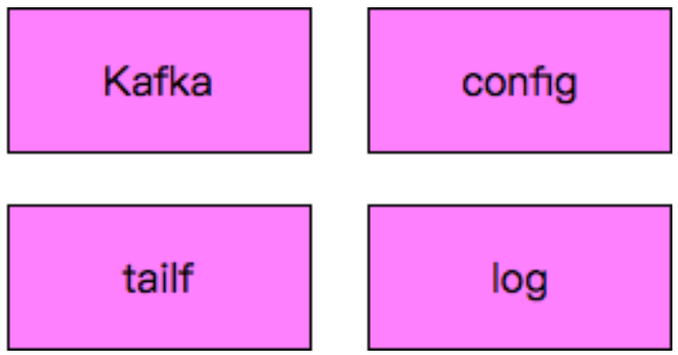
1）打开链接：http://kafka.apache.org/downloads.html下载kafka2.1.2

2）打开config目录下的server.properties， 修改log.dirs为D:\kafka_logs,
修改advertised.host.name=服务器ip

3）启动kafka ./bin/windows/kafka-server-start.bat ./config/server.preperties

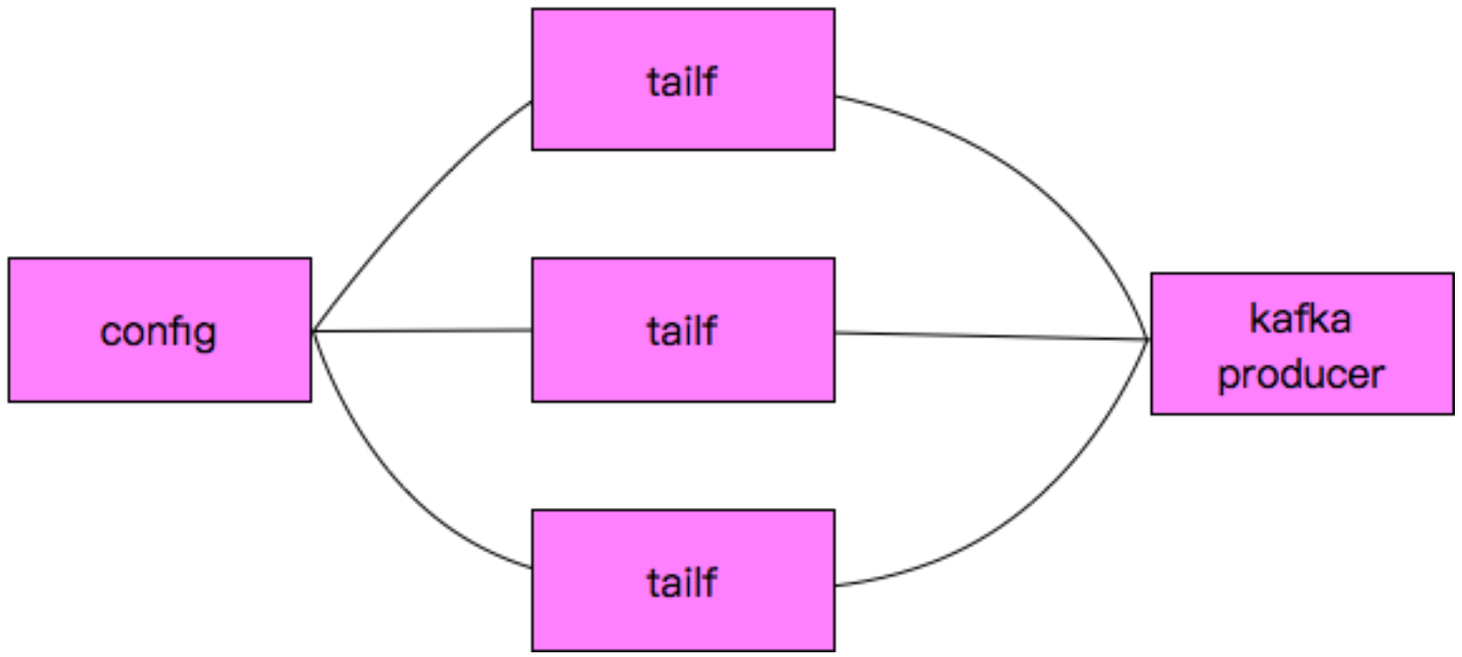
日志收集系统架构

9. log agent设计



日志收集系统架构

10. log agent流程



日志收集系统架构

11. kafka示例代码

Import “github.com/Shopify/sarama"

```
package main

import (
	"fmt"
	"github.com/Shopify/sarama"
)

func main() {
	config := sarama.NewConfig()
	config.Producer.RequiredAcks = sarama.WaitForAll
	config.Producer.Partitioner = sarama.NewRandomPartitioner
	config.Producer.Return.Successes = true

	msg := &sarama.ProducerMessage{}
	msg.Topic = "nginx_log"
	msg.Value = sarama.StringEncoder("this is a good test, my message is good")
	client, err := sarama.NewSyncProducer([]string{"192.168.31.177:9092"}, config)
	if err != nil {
		fmt.Println("producer close, err:", err)
		return
	}

	defer client.Close()

	pid, offset, err := client.SendMessage(msg)
	if err != nil {
		fmt.Println("send message failed,", err)
		return
	}

	fmt.Printf("pid:%v offset:%v\n", pid, offset)
}
```

日志收集系统架构

12. tailf组件使用

Import “github.com/hpcloud/tail”

```
package main

import (
	"fmt"
	"github.com/hpcloud/tail"
	"time"
)
func main() {
	filename := "./my.log"
	tails, err := tail.TailFile(filename, tail.Config{
		ReOpen:    true,
		Follow:    true,
		Location:  &tail.SeekInfo{Offset: 0, Whence: 2},
		MustExist: false,
		Poll:      true,
	})
	if err != nil {
		fmt.Println("tail file err:", err)
		return
	}
	var msg *tail.Line
	var ok bool
	for true {
		msg, ok = <-tails.Lines
		if !ok {
			fmt.Printf("tail file close reopen, filename:%s\n", tails.Filename)
			time.Sleep(100 * time.Millisecond)
			continue
		}
		fmt.Println("msg:", msg)
	}
}
```

日志收集系统架构

13. 配置文件库使用

Import “github.com/astaxie/beego/config”

1. 初始化配置库

```
iniconf, err := NewConfig("ini", "testini.conf")
if err != nil {
t.Fatal(err)
}
```

2. 读取配置项

- String(key string) string
- Int(key string) (int, error)
- Int64(key string) (int64, error)
- Bool(key string) (bool, error)
- Float(key string) (float64, error)

```
package main

import (
	"fmt"
	"github.com/astaxie/beego/config"
)

func main() {
	conf, err := config.NewConfig("ini", "./logcollect.conf")
	if err != nil {
		fmt.Println("new config failed, err:", err)
		return
	}

	port, err := conf.Int("server::port")
	if err != nil {
		fmt.Println("read server:port failed, err:", err)
		return
	}

	fmt.Println("Port:", port)
	log_level, err := conf.Int("log::log_level")
	if err != nil {
		fmt.Println("read log_level failed, ", err)
		return
	}
	fmt.Println("log_level:", log_level)

	log_path := conf.String("log::log_path")
	fmt.Println("log_path:", log_path)
}
```

日志收集系统架构

14. 日志库的使用

Import “github.com/astaxie/beego/logs”

1. 配置log组件

```
config := make(map[string]interface{}
config["filename"] = "./logs/logcollect.log"
config["level"] = logs.LevelDebug
configStr, err := json.Marshal(config)
if err != nil {
        fmt.Println("marshal failed, err:", err)
        return
}
```

2. 初始化日志组件

logs.SetLogger(“file”, string(configStr))

```
package main

import (
	"encoding/json"
	"fmt"
	"github.com/astaxie/beego/logs"
)

func main() {
	config := make(map[string]interface{})
	config["filename"] = "./logs/logcollect.log"
	config["level"] = logs.LevelDebug

	configStr, err := json.Marshal(config)
	if err != nil {
		fmt.Println("marshal failed, err:", err)
		return
	}

	logs.SetLogger(logs.AdapterFile, string(configStr))

	logs.Debug("this is a test, my name is %s", "stu01")
	logs.Trace("this is a trace, my name is %s", "stu02")
	logs.Warn("this is a warn, my name is %s", "stu03")
}
```

课后作业

1. 把今天的日志收集客户端，自己实现一遍