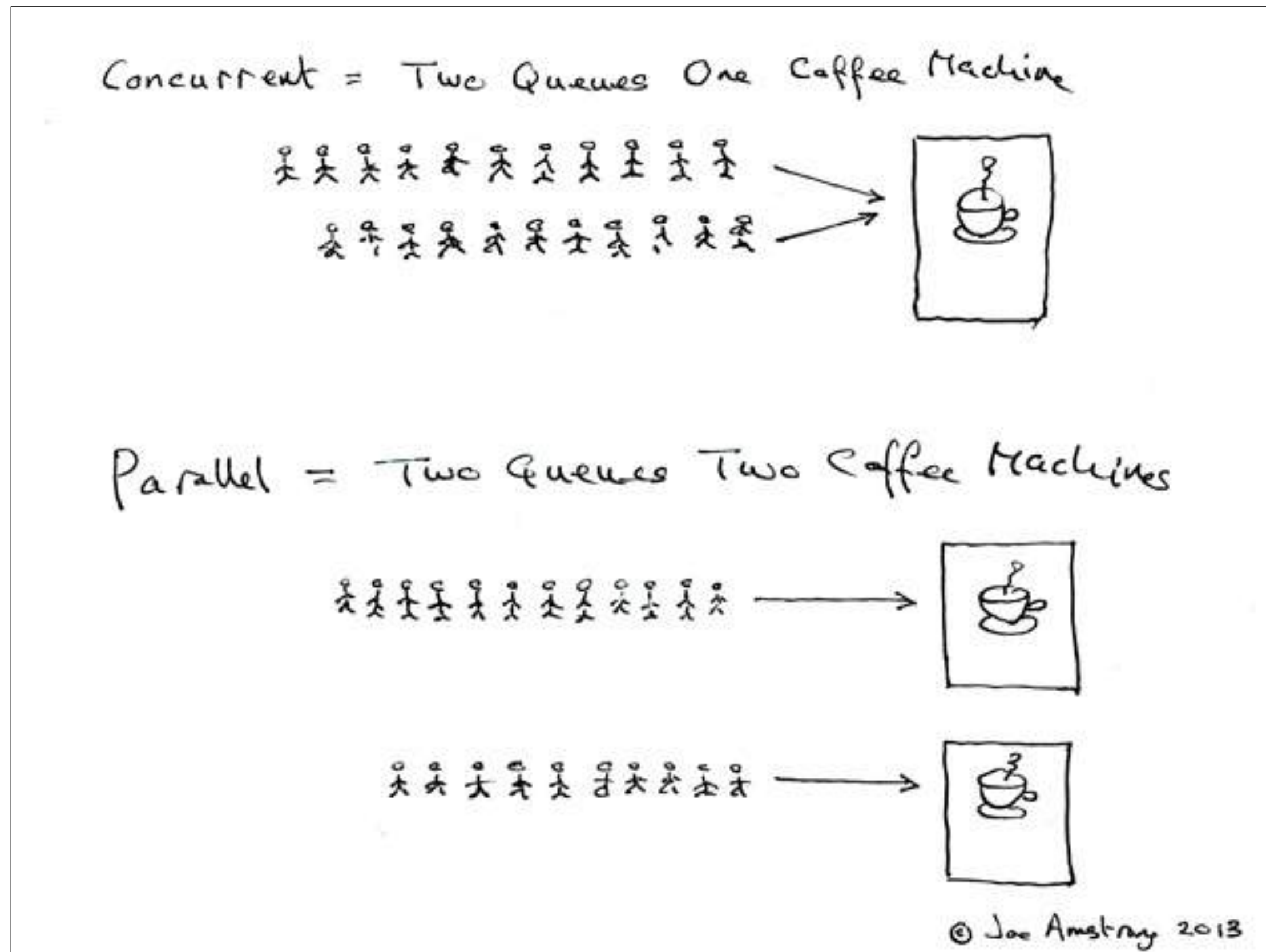


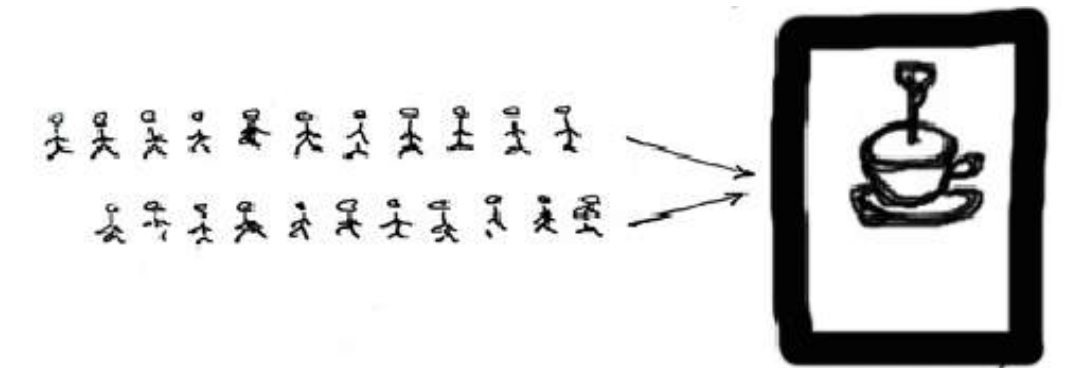
# 第三部分

# 流量应对策略

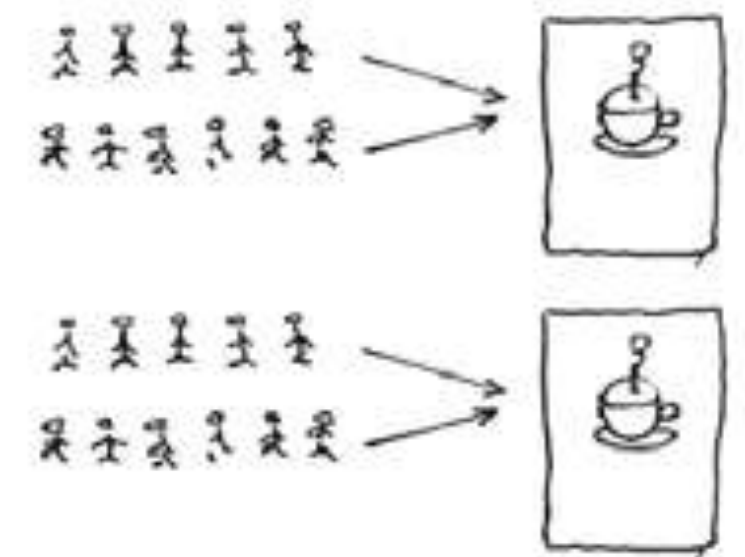
# 流量≈并发



• 更强的machine



• 更多machine(cap)



• 限流降级

Concurrency is about *dealing with* lots of things at once.  
Parallelism is about *doing* lots of things at once.

-----Joe Armstrong

# 关键点



CAP



分而治之，缩小竞争面

- 业务纵向拆分，化整为零
- 资源拆分，横向扩展



加速资源交换，更快响应

- cache , index , partition
- parallel
- non-blocking



共享资源串行操作  
数据一致性(脏读,丢失更新等)

- sync、lock , cas
- 额度、库存、积分、优惠券...

# 数据竞争

[sql方案示例]

数据库锁(全局标识拦截)：

```
update    product
  set     stock=stock-X
  where   stock-X >= 0
```

限流：

- max\_connections(db)
- max\_request+max\_threads(middleware)

- 乐观锁，带来重试代价
- 悲观锁，开销大，吞吐量差

# 数据竞争

## [NoSql方案示例]

### Redis+java方案（瑕疵版）：

```
stock=(incrby stock -X)
if (stock<0){incrby stock X;}
else { //submit}
```

### 限流、熔断：

- maxclients(redis)
- max\_request+max\_threads(middleware)
- hystrix..(service)

- 内存操作
- 单线程原子操作
- 高可用保障
- 兜底策略

# 数据竞争

## [NoSql方案示例2]

- 存储+运算，一致性保证
- 高可用措施保障
- 兜底策略

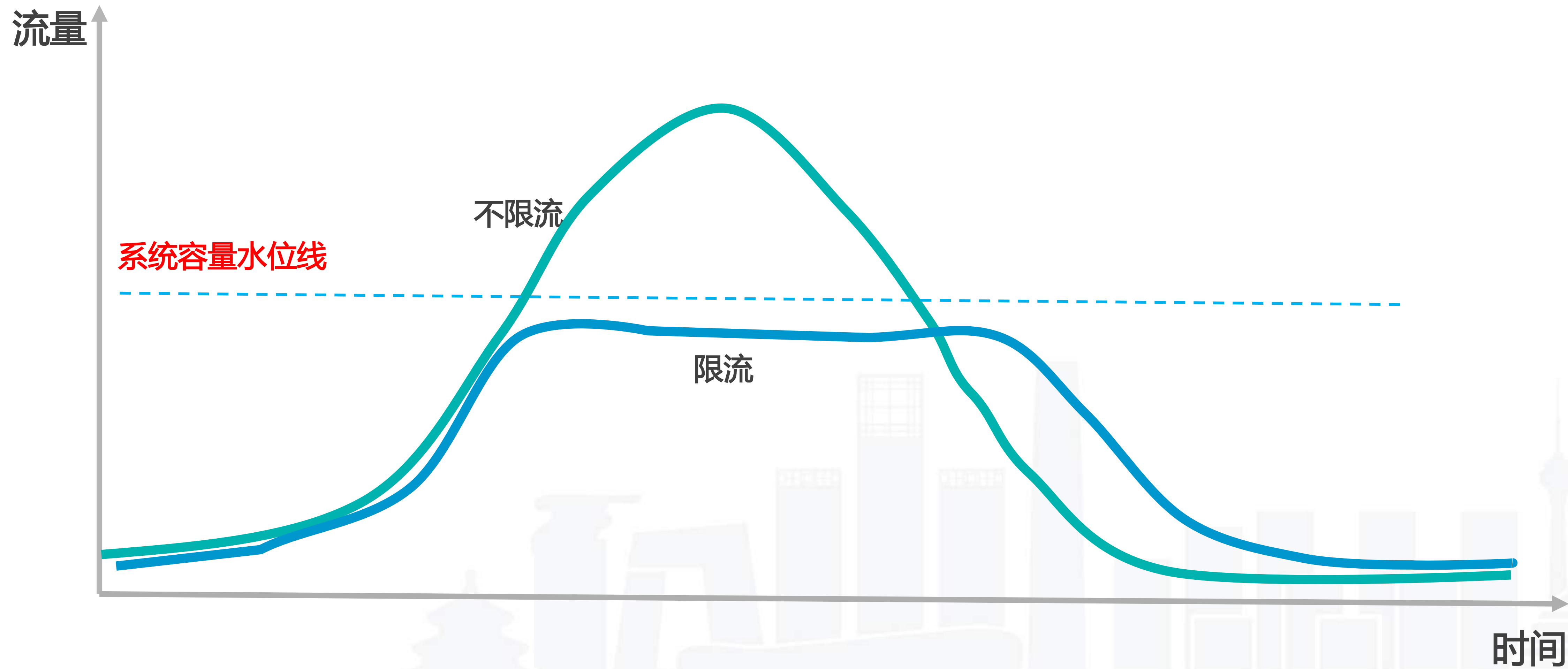
## Redis+(lua/module)方案：

```
local stock = redis.call("GET","STOCK")
if tonumber(stock)<tonumber(X) then
  return '-1'
else stock=redis.call("INCRBY", "STOCK",-X)
  return stock
end
```

## 限流、熔断：

- maxclients(redis)
- max\_request+max\_threads(middleware)
- hystrix..(service)

# 热点削峰



# 削峰策略 [知容量, 明底线]

底线

熔断

- 过载保护

丢车保帅

降级

- 拒绝服务
- 异步延迟

容量水位线

限流

- 多节点参与
- 开关控制



# 多节点有序参与

终端

- 本地缓存，防刷，流控

域名

- 智能DNS，DNS负载均衡

机房

- 多IDC，区域容灾，多ISP

LB / NG..

- 限流(limit, lua)，openresty，4层/7层LB

网关

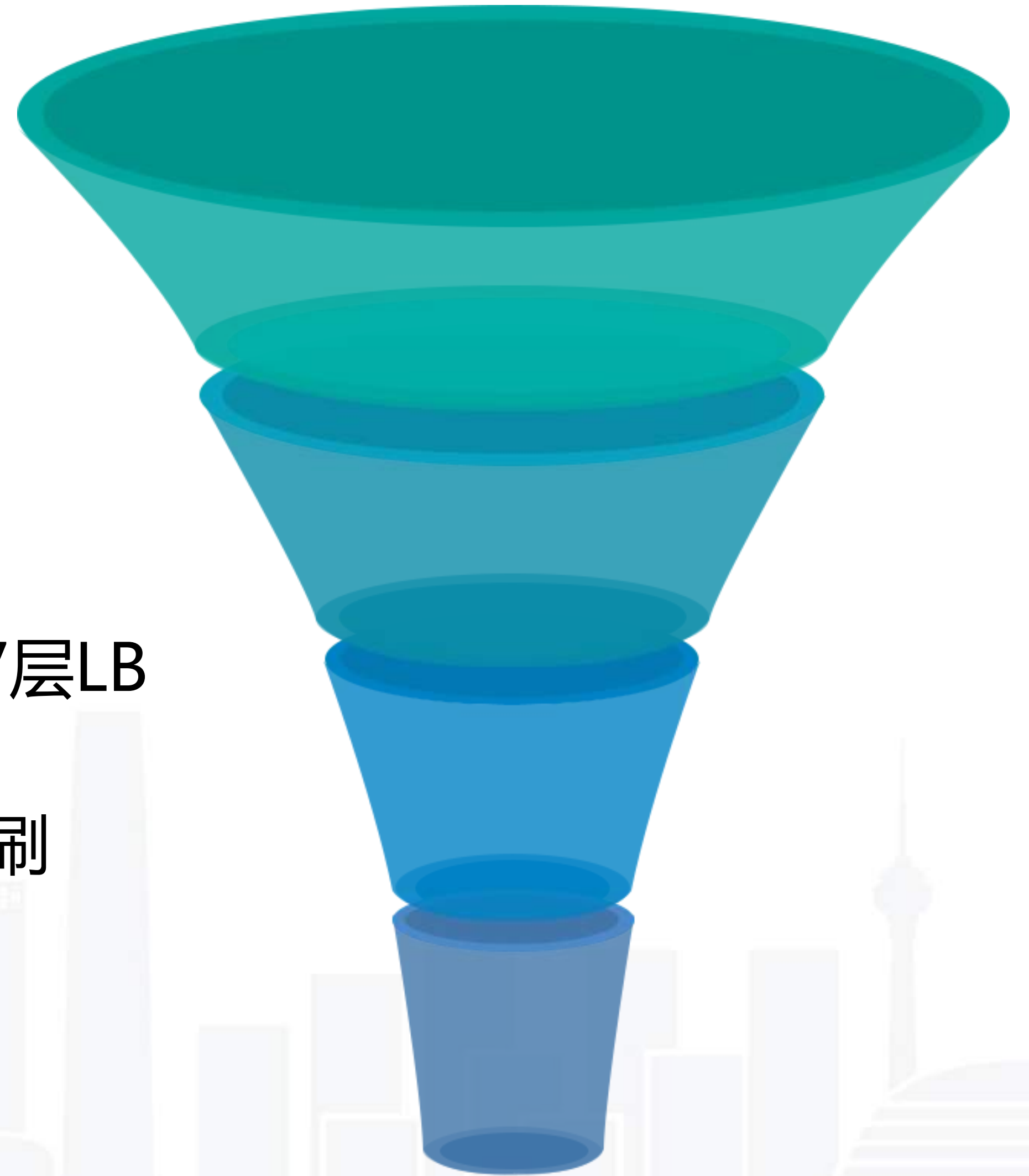
- 弹性扩容，限流(token)，熔断，防刷

Cache

- 集群，高可用，分片

服务

- 降级，熔断，弹性扩容



# 抓大不能放小 [细节决定成败]

SO	SI	E	O	P	YGC	YGCT	FGC	FGCT	GCT
1.80	0.00	1.49	52.38	25.48	980818	16447.565	7700	433.856	16881.421
69.60	0.00	57.83	56.87	25.48	980824	16447.785	7700	433.856	16881.640
0.00	100.00	77.26	69.57	25.48	980845	16448.388	7700	433.856	16882.244
62.58	0.00	39.90	16.59	25.48	980874	16449.112	7702	433.891	16883.003
45.06	0.00	23.21	30.90	25.48	980902	16449.778	7702	433.891	16883.669
100.00	0.00	0.00	41.97	25.48	980922	16450.324	7702	433.891	16884.215
0.00	71.15	62.32	58.97	25.48	980943	16450.958	7702	433.891	16884.849
0.00	84.04	86.80	74.38	25.48	980973	16451.655	7702	433.891	16885.546
0.00	83.18	71.37	25.03	25.48	980995	16452.287	7704	434.072	16886.360
81.82	0.00	82.68	38.30	25.48	981034	16453.078	7704	434.072	16887.150
0.00	73.61	18.58	55.51	25.48	981069	16453.864	7704	434.072	16887.936

GC

0	10	.89.79:8398	10.	.16.56:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:28007	10.	.16.36:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:6543	10.	.16.56:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:27670	10.	.16.36:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:60566	10.	.16.56:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:1718	10.	.16.56:80	CLOSE_WAIT	off (0.00/0/0)

TCP连接

```
[work@gw-online01 opt]$ netstat -ano|grep "16.56.*CLOSE_WAIT"|wc -l  
963
```

0	10	.89.79:3717	10.	.16.56:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:32752	10	.16.36:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:40748	10	.16.36:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:8080	10	.16.68:32253	TIME_WAIT	timewait (51.1)
0	10	.89.79:28184	10	.16.36:80	CLOSE_WAIT	off (0.00/0/0)
0	10	.89.79:62483	10.	.16.56:80	CLOSE_WAIT	off (0.00/0/0)

```
p-nio-8080-exec-200" #263 daemon prio=5 os_prio=0 tid=0x00007f109630e800  
ava.lang.Thread.State: WAITING (parking)  
  at sun.misc.Unsafe.park(Native Method)  
  - parking to wait for <0x00000006ce533488> (a java.util.concurrent.L  
  at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)  
  at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObj  
  at org.apache.http.pool.AbstractConnPool.getPoolEntryBlocking(Abstrac  
  at org.apache.http.pool.AbstractConnPool.access$200 (AbstractConnPool.  
  at org.apache.http.pool.AbstractConnPool$2.get (AbstractConnPool.java:  
  - locked <0x000000075d6584e0> (a org.apache.http.pool.AbstractConnPoo  
  at org.apache.http.pool.AbstractConnPool$2.get (AbstractConnPool.java:  
  at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.lease  
  at org.apache.http.impl.conn.PoolingHttpClientConnectionManager$1.get
```

线程阻塞 > 300

- 中间件内存管理、线程状态，连接状况
- db的io，慢sql，索引，join等
- 代码review，数据结构，日志

# 第四部分

## 关于监控

# 如果没有监控...



盲人骑瞎马，夜半临深池

# 监控体系

趋于个性



具有共性

应用/框架/业务逻辑/系统间调用



- 期望更轻量、无侵入性的业务监控
- cat , elk , zipkin等

中间件/缓存/数据库/代理/MQ...

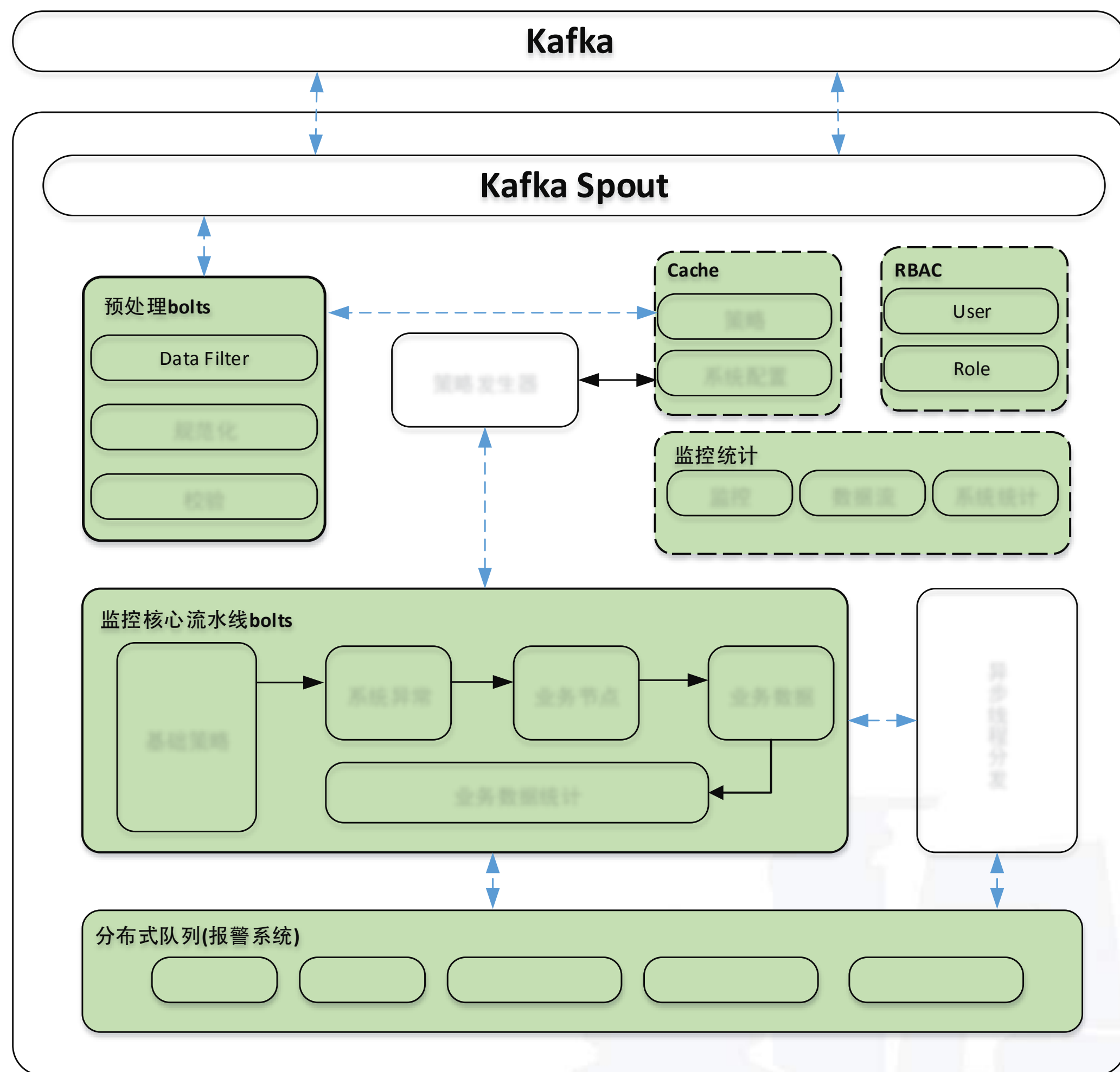


- zabbix , datagod, prometheus...
- apm工具 , 商业产品

OS/网络/存储/防火墙...



# 自研日志监控[轻量无侵入]



- 经典流式计算架构，流水线策略，线性扩展
- 高性能监控核心，灵活的监控策略
- 关键词模式、上下文模式、时间窗口模式等
- 轻量、高效、稳定，0侵入

# 日志监控平台



监控名称: (运营监控)提现系统故障  
 报警策略: 运营监控报警  
 监控系统: 理财APP-server

监控策略类型: 任务数据监控  
 监控策略状态: 有效  
 优先级: 2  
 是否实时处理: 是  
 是否下发: 否

监控发送内容: 请输入监控发送内容  
 接口URI: 请输入接口URI

延迟: 等于 0 ms后触发监控报警  
 监控策略: 0 秒 经过 0 次 等待 0 秒 报警

[修改](#)

监控名称: (运营监控)提现系统故障  
 报警策略: 运营监控报警  
 监控系统: 理财APP-server

监控策略类型: 节点数据监控  
 监控策略状态: 有效  
 优先级: 2  
 是否实时处理: 是  
 是否下发: 否

节点匹配内容step1: 请输入节点匹配内容  
 节点匹配内容分组step2: 请输入节点匹配内容分组  
 节点匹配内容分组数据step3: 请输入节点匹配内容分组数据

节点数据: 等于 0 触发监控报警  
 监控策略: 0 秒 经过 0 次 等待 0 秒 报警

## 系统报警通知

4月14日

微信

本次报警信息如下

系统名称: 借款-短信平台

报警时间: 2018-04-14

11:32:44---2018-04-14 11:32:44

报警次数: 887

报警策略: 借款短信平台异常

报警级别: 1

[相关日志](#), [点击下方\[详情\]查看](#)

[详情](#)

拒收此报警 1/ 3/ 5/ 9/ 12小时

[取消拒收](#) [修改报警组](#)

名称	详情
系统名称	借款-短信平台
机器IP地址	10.130.80.79
报警时间	2018-04-14 11:32:44---2018-04-14 11:32:44
报警次数	887
LOG_ID	
报警级别	1
报警策略名	借款短信平台异常
	2018-04-14 11:32:44,914 ERROR [com.creditease.sms.smserviceimpl.SMSInterfaceZUCPIImpl] - {} java.lang.NumberFormatException: For input string: "

**ERROR**

- 微信/邮件/短信
- 高可靠,高响应
- 高性能
- 灵活配置

# 谈点感想





# 感想

## 01 微服务≠spring cloud≠容器化≠RPC

工具/框架是手段而不是目的

## 02 优秀的系统=

适应性架构设计(指导)+超强工程能力(落地)

## 03 安于现状=走下坡路

提升团队整体工程能力，前瞻性改造

## 04 技术要紧贴业务，接地气

技术是手段而不是目的，生产力适应生产关系，技术业务相互促进共同发展

An aerial view of a city, overlaid with a semi-transparent blue layer. The blue layer features a network of glowing white lines and nodes, suggesting a digital or financial network. The city below is visible through the blue overlay, with some buildings and roads highlighted in a lighter blue. The overall tone is futuristic and technological.

# Thanks!