基于mongodb的秒级监控系统

——dds-inspector

陈柯任(花名:莫归) 2017.06.15

个人经历

陈柯任 阿里巴巴资深研发工程师

曾在大众点评从事Mongodb和MySQL相关运维和自动化研发

现主要致力于分布式存储和nosql数据库相关领域

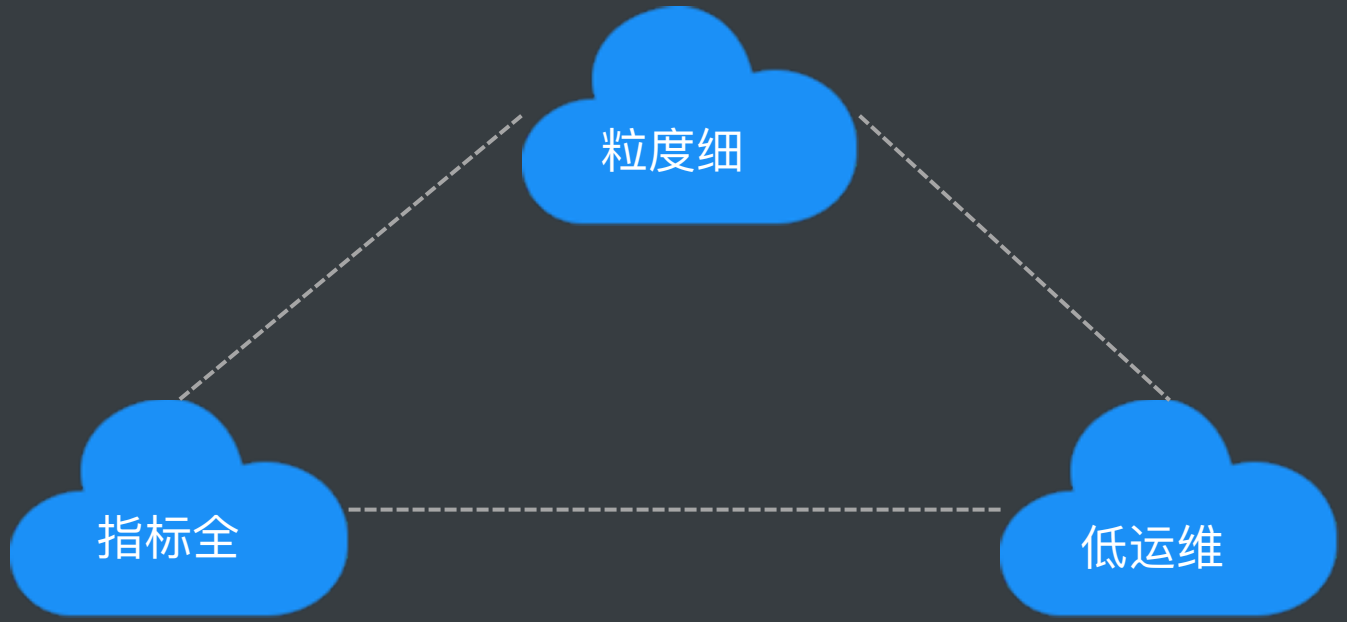
目录

- 设计目标
- 架构方案
- 数据模型
- 如何解决秒级采集带来的问题

目录

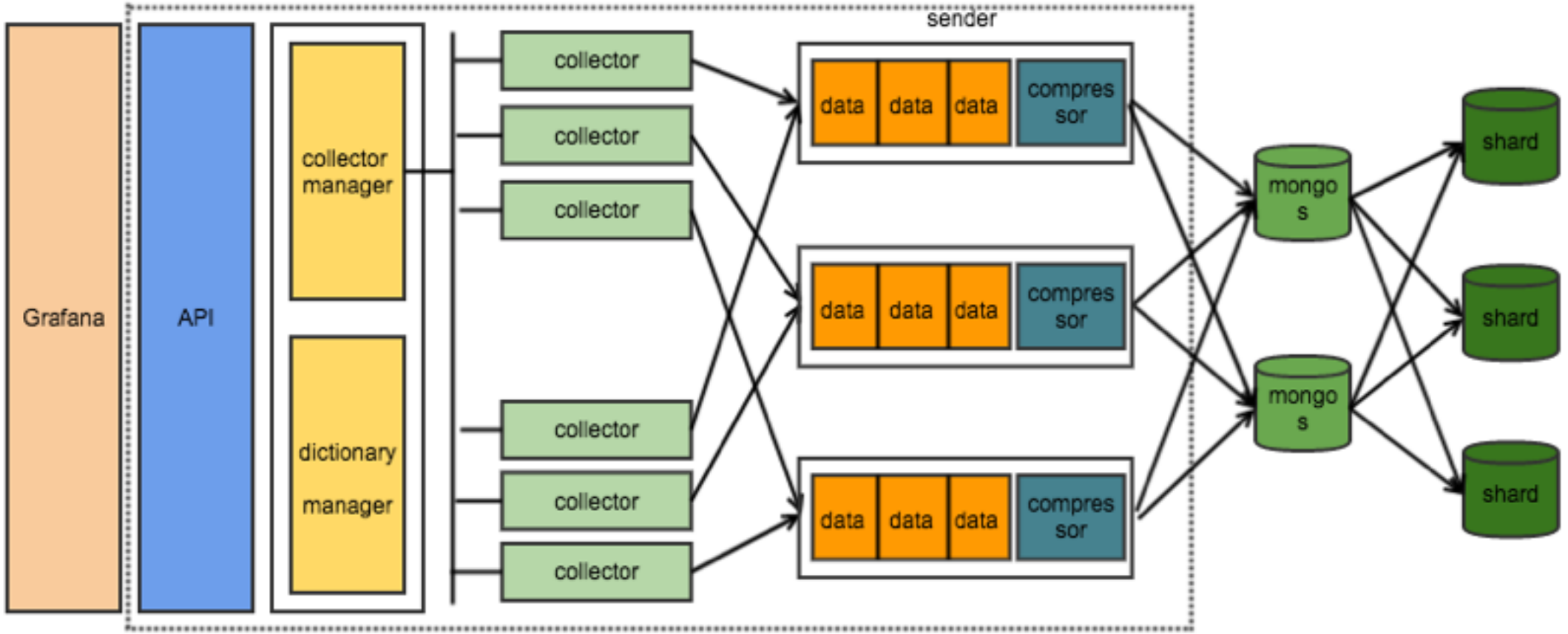
设计目标

我们希望有一个秒级粒度监控，全量指标数据采集，低运维成本的监控系统



目录

架构方案



Grafana + Go + Mongodb

架构方案

1. 1秒一个采集点
2. 全量数据指标，自动采集新指标
3. 单机多协程
4. 单核CPU可同时采集700+实例

目录

数据模型

- Key-Value方式存储

优点：

1. 灵活

缺点：

1. 数据大量冗余
2. 单个collection的document数太多
3. 索引大

。。。

{_id:1688,key:”command”,value:100,t:1497794541}
{_id:1688,key:”insert”,value:100,t:1497794541}
{_id:1688,key:”update”,value:200,t:1497794541}
{_id:1688,key:”cache_size”,value:132100,t:1497794541}
{_id:1688,key:”transaction”,value:128,t:1497794541}

数据模型

- 全量指标存一个文档

优点：

1. 指标聚合，document数减少

缺点：

1. 有效数据占比太低
2. document数依旧太多
3. 索引大

```
{
    _id:Object(123),
     hid:1688,
  time:1497794541
     inf:{
        insert:100,
        update:200,
        cache_size:134200,
        transaction:128,
        …
    }
}
```

```
{
    _id:Object(124),
     hid:1688,
  time:1497794542
     inf:{
        insert:100,
        update:200,
        cache_size:134200,
        transaction:128,
        …
    }
}
```

数据模型

- 一分钟时间数据聚合存储

优点：

1. 一分钟数据时间聚合，提高了有效数据率
2. document数量大大减少
3. 索引量减少，查询效率显著提升

缺点：

1. 查询时需要一点计算量

```
{
    _id:Object(123),
     hid:1688,
  t:1497794541
     inf:{
        insert:[100,105,201,230,…],
        update:[200,281,352,430,…],

cache_size:[134200,134200,134200,13
4200,…]
        transaction:[128,128,128,128,…]
        …
        pages evicted because they
exceeded the in-memory maximum:
[0,0,0,…]
    }
}
```

Shard key：{hid:1,t:1}

TTL：{t:1}

目录

主要问题

数据量大

频率高

数据压缩

- 将指标名自动进行编码，减小文档大小

```
{
   _id:Object(123),
    hid:1688,
 t:1497794541
    inf:{
       insert:[100,105,201,230,…],
       update:[200,281,352,430,…],

cache_size:[134200,134200,134200,134200,
…]
       transaction:[128,128,128,128,…]
       …
       pages evicted because they exceeded
the in-memory maximum:[0,0,0,…]
   }
}
```

+

```
{
   insert:”k1”,
   update:”k2”,
   cache_size:”k3”,
   trasaction:”k4”,
   …
   pages evicted because they
exceeded the in-memory maximum:”k5”
}
```

=

```
{
   _id:Object(123),
    hid:1688,
 t:1497794541
    inf:{
       k1:[100,105,201,230,…],
       k2:[200,281,352,430,…],
       k3:[134200,134200,134200,134200,…]
      k4:[128,128,128,128,…]
       …
      k5:[0,0,0,…]
   }
}
```

数据压缩

存储成本：

单进程:

1分钟存储量为：491k

一天存储量为：491 * 60 * 24 = 707040k ≈ 690M

一个月存储量为：690 * 30 = 20700M ≈ 20G

1w个进程实例：

一个月存储量为：20G * 10000 ≈ 195.3125T

网络流量：

假设一分钟采集1w个实例，且均匀分配网络IO：

167 * 491k = 81997k ≈ 81M

数据压缩

存储损耗点：

1. 大量重复指标数据
   如 transaction:[128,128,128…]

2. bson数据格式编码时针对数组有一个隐藏下标
   如:[1,2,3]实际bson存储格式为0:1,1:2,2:3

3. 所有数字都以int64，或者int32等大数据类型存储，浪费空间

4. 相邻数据之间变化不大，存本身不如存变化

数据压缩

编码格式：

原始数据

编码数据

flag（1bit）	意义
0	未用去重压缩
1	采用去重压缩

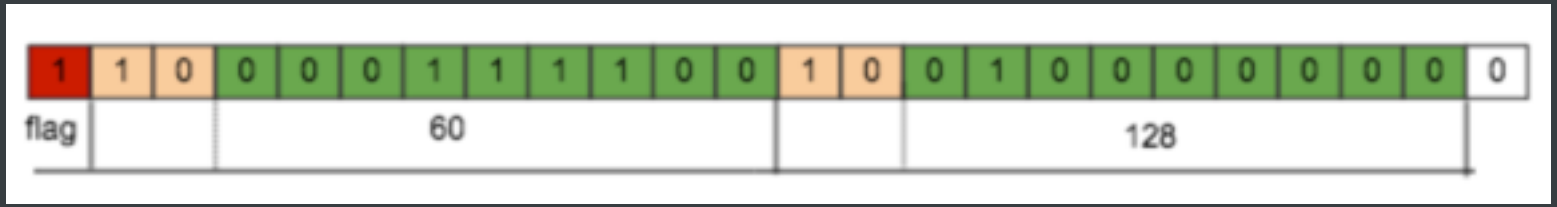
Flag(bit)	Value length	Value range	Total length(bit)
0	0	[0,0]	1
10	9	[-2^8+1,2^8-1]	11
110	12	[-2^11+1,2^11-1]	15
1110	64	[-2^63+1,2^63-1]	68
1111	0	null	4

数据压缩

例子1：

原始数据　　[128,128, 128,…,128] 长度为60

编码后数据　　[60,128]



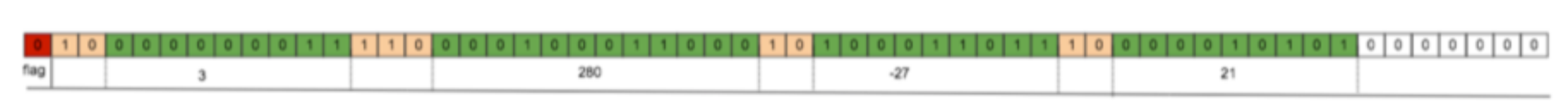
3Byte

数据压缩

例子2:

原始数据 [280,253,274]

编码后数据 [3,280,-27,21]



7Byte

数据压缩

压缩结果对比

纵向对比	压缩前	压缩后
单实例1分钟数据大小	491k	17.4k
1w实例一个月数据大小	195.3125T	7T(落盘5T)

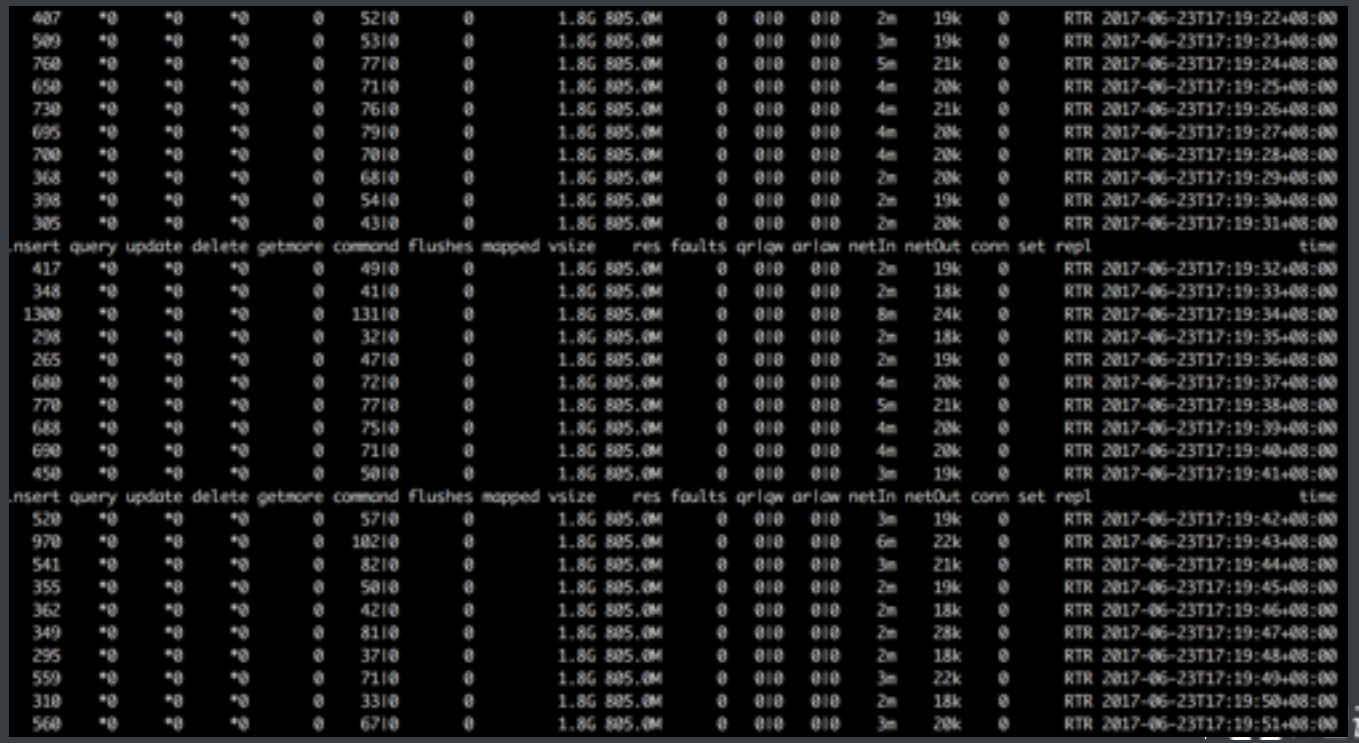
横向对比	压缩效果
influxdb	each point down to around 2.2 bytes per point
dds-inspector	17.4k / 711 / 60 ≈ 0.4byte

流量削峰

1. 多个send协程发送，每个协程一个发送队列
2. 固定个数/固定间隔时间发送数据
2. 1分钟间隔不以时钟的整分来计，而是以采集协程启动时开始计算

结果：

流量几乎均匀分配在一分钟时间里



insert	query	update	delete	getmore	command	flushes	mapped	vsize	res	faults	qr|qw	ar|aw	netIn	netOut	conn	set	repl	time
407	*0	*0	*0	0	521|0	0	1.8G	805.0M	0	0|0	0|0	2m	19k	0		RTR	2017-06-23T17:19:22+08:00
509	*0	*0	*0	0	533|0	0	1.8G	805.0M	0	0|0	0|0	3m	19k	0		RTR	2017-06-23T17:19:23+08:00
760	*0	*0	*0	0	771|0	0	1.8G	805.0M	0	0|0	0|0	5m	21k	0		RTR	2017-06-23T17:19:24+08:00
650	*0	*0	*0	0	711|0	0	1.8G	805.0M	0	0|0	0|0	4m	20k	0		RTR	2017-06-23T17:19:25+08:00
730	*0	*0	*0	0	761|0	0	1.8G	805.0M	0	0|0	0|0	4m	21k	0		RTR	2017-06-23T17:19:26+08:00
695	*0	*0	*0	0	791|0	0	1.8G	805.0M	0	0|0	0|0	4m	20k	0		RTR	2017-06-23T17:19:27+08:00
700	*0	*0	*0	0	701|0	0	1.8G	805.0M	0	0|0	0|0	4m	20k	0		RTR	2017-06-23T17:19:28+08:00
368	*0	*0	*0	0	681|0	0	1.8G	805.0M	0	0|0	0|0	2m	20k	0		RTR	2017-06-23T17:19:29+08:00
398	*0	*0	*0	0	541|0	0	1.8G	805.0M	0	0|0	0|0	2m	19k	0		RTR	2017-06-23T17:19:30+08:00
305	*0	*0	*0	0	431|0	0	1.8G	805.0M	0	0|0	0|0	2m	20k	0		RTR	2017-06-23T17:19:31+08:00
insert	query	update	delete	getmore	command	flushes	mapped	vsize	res	faults	qr|qw	ar|aw	netIn	netOut	conn	set	repl	time
417	*0	*0	*0	0	491|0	0	1.8G	805.0M	0	0|0	0|0	2m	19k	0		RTR	2017-06-23T17:19:32+08:00
348	*0	*0	*0	0	411|0	0	1.8G	805.0M	0	0|0	0|0	2m	18k	0		RTR	2017-06-23T17:19:33+08:00
1300	*0	*0	*0	0	1311|0	0	1.8G	805.0M	0	0|0	0|0	8m	24k	0		RTR	2017-06-23T17:19:34+08:00
298	*0	*0	*0	0	321|0	0	1.8G	805.0M	0	0|0	0|0	2m	18k	0		RTR	2017-06-23T17:19:35+08:00
265	*0	*0	*0	0	471|0	0	1.8G	805.0M	0	0|0	0|0	2m	19k	0		RTR	2017-06-23T17:19:36+08:00
680	*0	*0	*0	0	721|0	0	1.8G	805.0M	0	0|0	0|0	4m	20k	0		RTR	2017-06-23T17:19:37+08:00
770	*0	*0	*0	0	771|0	0	1.8G	805.0M	0	0|0	0|0	5m	21k	0		RTR	2017-06-23T17:19:38+08:00
688	*0	*0	*0	0	751|0	0	1.8G	805.0M	0	0|0	0|0	4m	20k	0		RTR	2017-06-23T17:19:39+08:00
690	*0	*0	*0	0	711|0	0	1.8G	805.0M	0	0|0	0|0	4m	20k	0		RTR	2017-06-23T17:19:40+08:00
450	*0	*0	*0	0	501|0	0	1.8G	805.0M	0	0|0	0|0	3m	19k	0		RTR	2017-06-23T17:19:41+08:00
insert	query	update	delete	getmore	command	flushes	mapped	vsize	res	faults	qr|qw	ar|aw	netIn	netOut	conn	set	repl	time
520	*0	*0	*0	0	571|0	0	1.8G	805.0M	0	0|0	0|0	3m	19k	0		RTR	2017-06-23T17:19:42+08:00
970	*0	*0	*0	0	1021|0	0	1.8G	805.0M	0	0|0	0|0	6m	22k	0		RTR	2017-06-23T17:19:43+08:00
541	*0	*0	*0	0	821|0	0	1.8G	805.0M	0	0|0	0|0	3m	21k	0		RTR	2017-06-23T17:19:44+08:00
355	*0	*0	*0	0	501|0	0	1.8G	805.0M	0	0|0	0|0	2m	19k	0		RTR	2017-06-23T17:19:45+08:00
362	*0	*0	*0	0	421|0	0	1.8G	805.0M	0	0|0	0|0	2m	18k	0		RTR	2017-06-23T17:19:46+08:00
349	*0	*0	*0	0	811|0	0	1.8G	805.0M	0	0|0	0|0	2m	28k	0		RTR	2017-06-23T17:19:47+08:00
295	*0	*0	*0	0	371|0	0	1.8G	805.0M	0	0|0	0|0	2m	18k	0		RTR	2017-06-23T17:19:48+08:00
559	*0	*0	*0	0	711|0	0	1.8G	805.0M	0	0|0	0|0	3m	22k	0		RTR	2017-06-23T17:19:49+08:00
310	*0	*0	*0	0	331|0	0	1.8G	805.0M	0	0|0	0|0	2m	18k	0		RTR	2017-06-23T17:19:50+08:00
560	*0	*0	*0	0	671|0	0	1.8G	805.0M	0	0|0	0|0	3m	20k	0		RTR	2017-06-23T17:19:51+08:00

其它注意点

1. 使用对象池减少内存使用和GC消耗
2. batch insert 而不要使用upsert
2. Mgo的bson反序列化时setterStyles在高并发下有大量锁争用，需自行优化
3. DB + Memory 数据访问API保证可查询问当前最新数据


MongoDB
中文社区
IT大咖说
知识分享平台
为了无法计算的价值
阿里云