**Cassandra调研**

**纪江霖**

*TP-LINK TECHNOLOGIES CO., LTD*

[*http://www.tp-link.com.cn*](http://www.tp-link.com.cn)

*SouthBuilding, No.5 Keyuan Road, Central Zone*

*Science & TechnologyPark, NanShan*

*ShenZhen, P. R. China*

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本/状态** | **作者** | **起止日期** | **描述** |
| V0.1 | 纪江霖 | 2015-03-25 |  |

# 摘要

**关键字：**

# NoSQL数据库入门

Nosql数据库多的让人眼花缭乱，要先掌握大的道，再去看具体的术。首先快速阅读了**《大规模分布式存储系统：原理解析与架构实战》**的前几章，了解了分布式存储系统的特点和基本原理。这本书是一个阿里巴巴技术专家写的，参与过OceanBase的开发，对于理解概念有一点帮助，但没有具体的细节。

快速浏览了《**Cassandra权威指南**》，这本书比较老了，但是，对于理解概念有点帮助。

1. **为什么使用Nosql数据库？**
2. 因为传统的数据库不适应互联网公司的数据的存储特点，关系数据库结构有固定的schema。Nosql表的结构灵活，可以随意增改。
3. 传统的数据库分区复杂，对于水平扩展（增加节点，改善性能）有局限性。
4. Nosql数据库不需要复杂的关系查询（比如join连接操作）。
5. **为什么使用Nosql数据库来搭建集群？**

节约成本，自动化部署，减少人工干预，降低运维成本。使用一堆硬件层面上廉价，低端的，容易出错的机器，在软件层面上实现高可用性，可扩展性，健壮的容错系统。

1. **参考资料：**
2. <http://cassandra.apache.org/>
3. <<Efficient Reconciliation and Flow Control for Anti-Entropy Protocols>> (未看)
4. Dynamo: Amazon's Highly Available Key-value Store
5. Bigtable: A Distributed Storage System for Structured Data
6. 《大规模分布式存储系统：原理解析与架构实战》
7. 《Cassandra权威指南》
8. <http://teddymaef.github.io/learncassandra/cn/replication/turnable_consistency.html>
9. <https://www.ibm.com/developerworks/cn/opensource/os-cn-cassandraxu2/>（cassandra存储的数据结构，未看）

# Cassandra简介

Apache Cassandra是一套开源分布式**NoSQL**数据库系统，采用**java**编写，可以在<http://cassandra.apache.org/>上下载它的源码来看。（**接下来有时间阅读源码，公司的内网很坑，很多链接打不开，很多参考资料在**<http://www.datastax.com/>上面，但是打不开链接**）**。

Cassandra集Google BigTable的数据模型与Amazon Dynamo的完全分布式的架构于一身。



Cassandra的类Dynamo特性有以下几点：

对称的，P2P架构

无特殊节点，无单点故障

基于Gossip的分布式管理

通过分布式hash表放置数据

类BigTable特性：

列族数据模型

SSTable磁盘存储

Append-only commit log

Memtable (buffer and sort)

不可修改的SSTable文件

# Cassandra数据模型

可以把cassandra看作是一个5维的稀疏的hash表:

**[Keyspace][ColumnFamily][Key][SuperColumn][SubColumn]**

**Keyspace**：Cassandra中最顶层的容器，可以看作关系数据库中的数据库。

**Row Key** ：是每一行的唯一标识。

**Column:** Cassandra 中存储的最小单位，key/value 对。Key和Value都可以是二进制。Cassandra取出的数据顺序是总是一定的，数据保存时已经按照定义的规则存放，所以取出来的顺序已经确定了。另外，Cassandra按照column name而不是column value来进行排序

用一个JSON来表示一下：

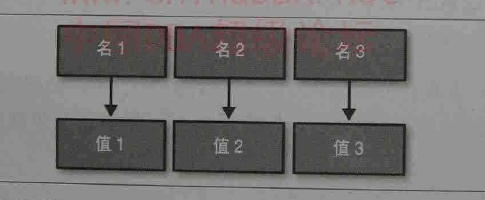
**{  // 这是一个Column**

**name: "emailAddress",**

**value: "arin@example.com",**

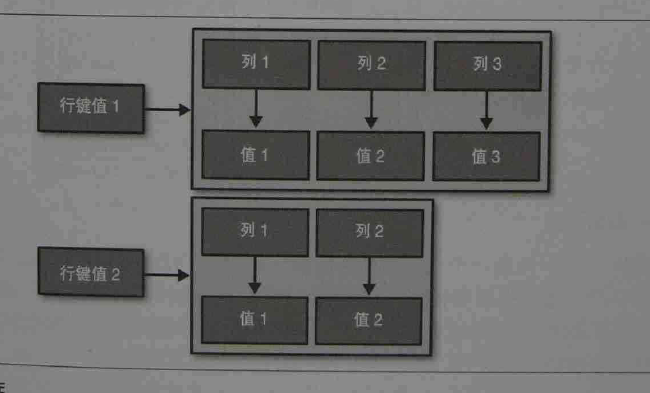
**timestamp: 123456789**

**}**



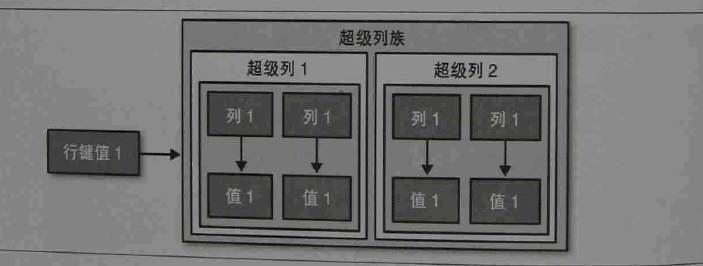
**Column**

**Column Family:** 可以理解成关系数据库中的一张表。有很多行，每一行由一个row key来唯一标识。每一个行是一个column的集合。如下图所示：



**Colum Family**

**SuperColumn ：由多个Column组成的。如图所示：**

****

**SuperColum / SuperColum Family**

# Cassandra分区策略



Cassandra中的**数据分布，数据路由，数据迁移**都使用上图中的环的模型。

如图所示，Cassandra使用一致性hash，把每个节点做hash，分布到一个环上。把每个key做一个hash，分布到一个环上。整个hash空间是一个环。按照顺时针的方向，也就是递增的方向，找到第一个值比hash（key）大的节点，把该key的数据放置在该节点上面。

采用这种hash的优势在于：**每个节点的增加和删除，只会影响到邻居节点**。

但是，单纯使用这个模型，会导致一些问题。因为每个节点的处理能力不一样，负载不一样，加入和删除节点时，可能导致数据分布不平衡。引入虚拟节点，使得原先的一个物理节点在环上对应更多的**虚拟节点**，保证数据分布平衡。

Cassandra中的每一个节点都可以当作一个代理，因为它知道每一个节点的地址和存活状态（具体是采用gossip协议来发现节点和检测故障的）。客户端可以直接连接到环上的任一节点上面，只需要**一跳**的代价，就可以找到目标存储的节点。

具体实现这种hash思想的有两种分区器：**Murmur3Partitioner，RandomPartitioner**。

ByteOrderedPartitioner分区器不是使用hash算法，它使用字符串进行排序，优点是可以使用范围查询，缺点显而易见，没有使用hash，分布不均匀，负载均衡困难。

Cassandra使用3种分区器。

1. Murmur3Partitioner（默认）：基于MurmurHash哈希算法

2 .RandomPartitioner：基于MD5哈希算法

3. ByteOrderedPartitioner：根据partition key的bytes进行有序分区。

# Cassandra故障检测

Gossip主要是用**来发现节点，发现节点的故障**。

Cassandra构建的集群，每个节点地位完全相同，没有中心，每个可以完成任意的路由（因为每个节点能够知道其他节点，并且知道其他节点的死活），任意节点的挂掉，都不会造成灾难性的整体影响，所以叫做没有单点失效。（spof）

Gossip就是一个**流言传播协议，病毒传播协议**，可以把信息以**指数**的速率传播出去。可以非常简单的证明，只需要LogN的时间传播一个节点的消息。（N是集群中节点的数目）

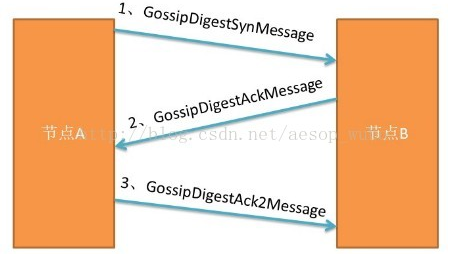
## |Gossip传播时间复杂度

如下图所示，假设节点A要在N个点中传播自己的状态，在第1个时间单位传播数目为2，在第2个时间单位传播数目为4…依次类推，是一个指数传播模式，所以有：T=logN

Gossip 协议传播示意图

所以，在同一时间中，每个节点随机的和其他节点交换信息，只需要logN的时间就可以发现集群中所有的其他的节点。

## |Gossip节点消息交换



这里假设192.168.1.1（源节点）决定和192.168.1.2（目标节点）同步，首先源节点向目标节点发送 GossipDigestSynMessage包，这个包包括本机维护的所有节点的状态信息的最新版本摘要，摘要只包含key和version，不包含具 体的value，这样可以减小同步的带宽消耗。

当目标节点收到GossipDigestSynMessage包时，它需要做两件事：

1. 找出收到的消息中比本地版本新的状态，按照版本号差异大小排序，将这些状态的摘要 放入 GossipDigestAckMessage中
2. 找出本地比源节点版本更新的状态，将这些状态 放入GossipDigestAckMessage

当GossipDigestAckMessage构建完成后，会被发送给源节点

这里按照版本号差异大小排序的原因是每个Message允许发送的状态数量是有限的，这样可以保证比较老的状态（版本号差异大的）可以优先得到更新。

源机器接收到GossipDigestAckMessage后，使用发送过来的目标节点更新的状态更新本地的状态，这样源节点就获取到了目标节点上 比自己更新的状态。

同时源节点把包含在GossipDigestAckMessage中摘要对应的状态通过GossipDigestAck2Message发送到目标 服务器，目标服务器更新本地的状态，这样目标服务器也获取到了源节点上比自己更新的状态。

完成这样一次同步后，源节点和目标节点上的状态都得到了同步**（交换的过程中包含了token，用于路由）。**

## |Gossip节点初始启动

当一个节点启动时，获取配置文件（storage-conf.xml）中的seeds配置，从而知道集群中所有的seed节点。

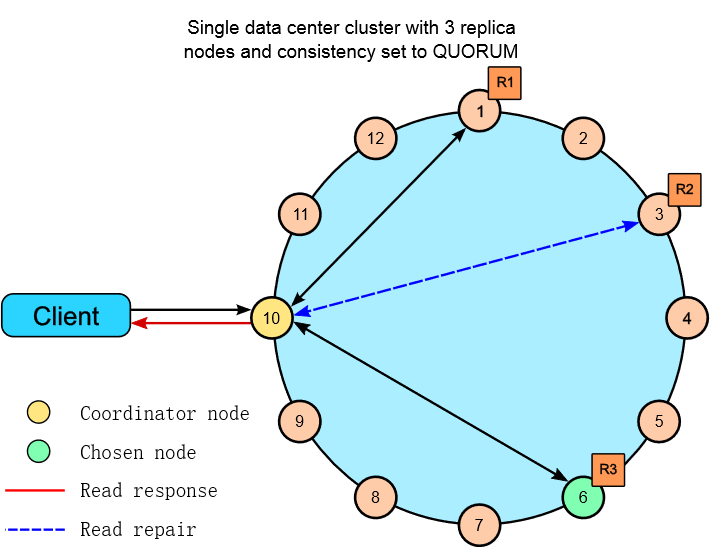
按照以下规则向其他节点发 送同步消息：

1. 随机取一个当前活着的节点，并向它发送同步请求。
2. 向随机一台不可达的机器发送同步请求。
3. 如果第一步中所选择的节点不是seed，或者当前活着的节点数少于seed数，则向随意一台seed发送同步请求。

# Cassandra 一致性

## 写操作的Consistency Level





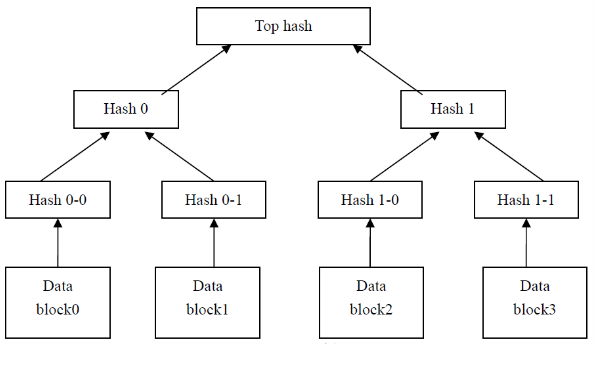
## 读操作的Consistency Level



1. 参见：<http://www.datastax.com/documentation/cassandra/2.0/cassandra/architecture/architectureDataDistributeReplication_c.html>（内网打不开）

## 逆熵

这是一种备份之间的同步机制。节点之间定期互相检查数据对象的一致性，这里采用的检查不一致的方法是 Merkle Tree。



## 读修复

客户端读取某个对象的时候，触发对该对象的一致性检查：

读取Key A的数据时，系统会读取Key A的所有数据副本，如果发现有不一致，则进行一致性修复。

如果读一致性要求为ONE，会立即返回离客户端最近的一份数据副本。然后会在后台执行Read Repair。这意味着第一次读取到的数据可能不是最新的数据；如果读一致性要求为QUORUM，则会在读取超过半数的一致性的副本后返回一份副本给客户端，剩余节点的一致性检查和修复则在后台执行；如果读一致性要求高(ALL)，则只有Read Repair完成后才能返回一致性的一份数据副本给客户端。可见，该机制有利于减少最终一致的时间窗口。

## 移交提示

对写操作，如果其中一个目标节点不在线，先将该对象中继到另一个节点上，中继节点等目标节点上线再把对象给它：

Key A按照规则首要写入节点为N1，然后复制到N2。假如N1宕机，如果写入N2能满足ConsistencyLevel要求，则Key A对应的RowMutation将封装一个带hint信息的头部（包含了目标为N1的信息），然后随机写入一个节点N3，此副本不可读。同时正常复制一份数据到N2，此副本可以提供读。如果写N2不满足写一致性要求，则写会失败。 等到N1恢复后，原本应该写入N1的带hint头的信息将重新写回N1。

## 删除

单机删除非常简单，只需要把数据直接从磁盘上去掉即可，而对于分布式，则不同，分布式删除的难点在于：如果某对象的一个备份节点 A 当前不在线，而其他备份节点删除了该对象，那么等 A 再次上线时，它并不知道该数据已被删除，所以会尝试恢复其他备份节点上的这个对象，这使得删除操作无效。Cassandra 的解决方案是：本地并不立即删除一个数据对象，而是给该对象标记一个hint，定期对标记了hint的对象进行垃圾回收。在垃圾回收之前，hint一直存在，这使得其他节点可以有机会由其他几个一致性保证机制得到这个hint。Cassandra 通过将删除操作转化为一个插入操作，巧妙地解决了这个问题。

# Cassandra 存储机制

Cassandra的存储机制借鉴了Bigtable的设计，采用Memtable和SSTable的方式。

## 7.1 CommitLog

Cassandra在写数据之前，也需要先记录日志，称之为Commit Log，然后数据才会写入到Column Family对应的MemTable中，且MemTable中的数据是按照key排序好的。SSTable一旦完成写入，就不可变更，只能读取。下一次Memtable需要刷新到一个新的SSTable文件中。所以对于Cassandra来说，可以认为只有顺序写，没有随机写操作。

## 7.2 MemTable

MemTable是一种内存结构，当数据量达到块大小时，将批量flush到磁盘上，存储为SSTable。这种机制，相当于缓存写回机制(Write-back Cache)，优势在于将随机IO写变成顺序IO写，降低大量的写操作对于存储系统的压力。所以我们可以认为Cassandra中只有顺序写操作，没有随机写操作。

## 7.3 SSTable

SSTable是Read Only的，且一般情况下，一个CF会对应多个SSTable，当用户检索数据时，Cassandra使用了Bloom Filter，即通过多个hash函数将key映射到一个位图中，来快速判断这个key属于哪个SSTable。

为了减少大量SSTable带来的开销，Cassandra会定期进行compaction，简单的说，compaction就是将同一个CF的多个SSTable合并成一个SSTable。在Cassandra中，compaction主要完成的任务是：

1）垃圾回收： cassandra并不直接删除数据，因此磁盘空间会消耗得越来越多，compaction 会把标记为删除的数据真正删除；

2）合并SSTable：compaction 将多个 SSTable 合并为一个（合并的文件包括索引文件，数据文件，bloom filter文件），以提高读操作的效率；

3）生成 MerkleTree：在合并的过程中会生成关于这个 CF 中数据的 MerkleTree，用于与其他存储节点对比以及修复数据。

## 7.4 压缩

为了减少大量SSTable带来的开销，Cassandra会定期进行compaction，简单的说，compaction就是将同一个CF的多个SSTable合并成一个SSTable。在Cassandra中，compaction主要完成的任务是：

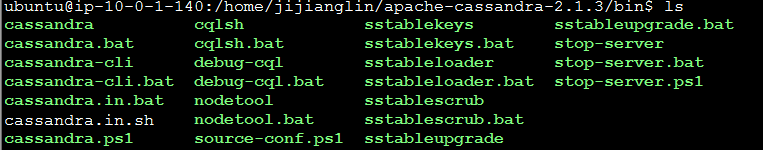
1）垃圾回收： cassandra并不直接删除数据，因此磁盘空间会消耗得越来越多，compaction 会把标记为删除的数据真正删除；

2）合并SSTable：compaction 将多个 SSTable 合并为一个（合并的文件包括索引文件，数据文件，bloom filter文件），以提高读操作的效率；

3）生成 MerkleTree：在合并的过程中会生成关于这个 CF 中数据的 MerkleTree，用于与其他存储节点对比以及修复数据。

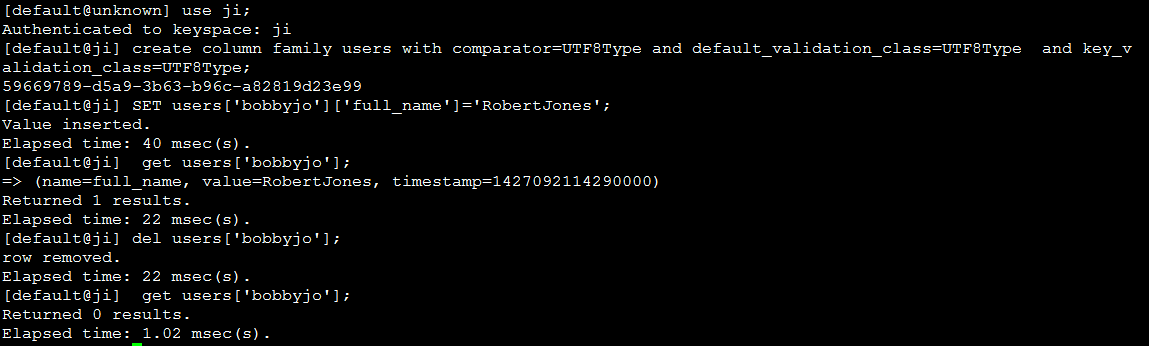
# Cassandra 服务端

在aws server上面安装java jdk 1.7，启动Cassandra ：./Cassandra &后台执行，或者使用-f 在前台执行）



如图，cassandra是后台服务，cassandra-cli和cqlsh是客户端CLI， cassandra-cli在3.0的版本里面将会被废弃。

使用cassandra-cli进行操作，如下示意图：



Cqlsh是官方推荐的操作方式，类似关系数据库中的sql语句，可以进行CURD(增删查改的操作)

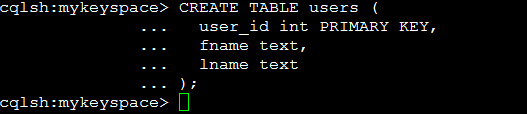
创建表空间：



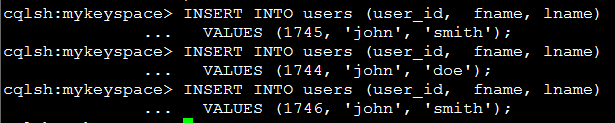
使用表空间：



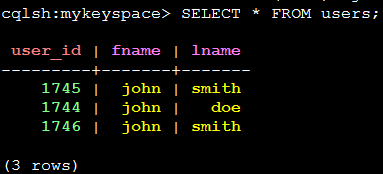
创建表：



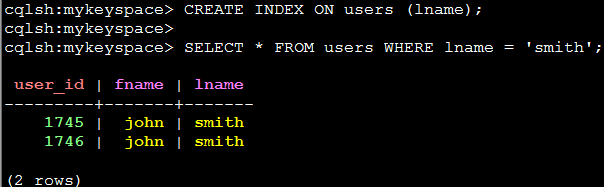
插入数据：



查询数据：



创建索引，查找数据：



# Cassandra写性能测试

**（1）**、在亚马逊的3台机器上面搭建集群，这3台机器构成一个后台的server。（集群之前使用gossip通信，是p2p模式，每个节点地位平等，没有特殊节点，但是有有种子节点）

1. 种子节点：ec2-54-223-174-20.cn-north-1.compute.amazonaws.com.cn，局域网地址：10.0.1.140
2. ec2-54-223-180-186.cn-north-1.compute.amazonaws.com.cn 局域网地址：10.0.1.187
3. ec2-54-223-181-182.cn-north-1.compute.amazonaws.com.cn 局域网地址：10.0.1.186

设置节点的listen\_address 和rpc\_address. rpc\_address用于客户端连接，

listen\_address供节点之间通信使用。我们把listen\_address和rpc\_address都设为本机

IP地址。

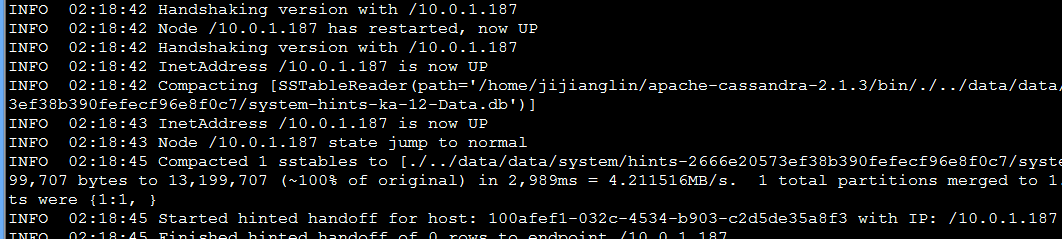




设置非种子节点的seed，seed是用来感知其他节点的。



启动后节点进入normal 状态。



**（2）**、客户端部署在ec2-54-223-181-190.cn-north-1.compute.amazonaws.com.cn 局域网地址：10.0.1.188。使用在myeclipse上面把客户端程序打成一个jar包，放在10.0.1.188上面运行。

Java –jar xxx.jar 50 10000 –Xmx 1024m ,测试TPS（50代表50个线程，10000代表每个线程做10000个操作）。参见本目录中的myeclipse工程源代码。主函数在TestSimpleClient.java 里面。

**（3）**、集群机器配置。

|  |  |
| --- | --- |
| 参数 | 值 |
| 服务端机器CPU | 1核Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz |
| 服务端机器 内存 | 总共992M，使用了287M，剩余704M |
| 客户端机器 CPU | 1核Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz |
| 客户端机器 内存 | 总共992M，使用了79M，剩余913M |
| 客户端 | 使用datastax提供的driver |

使用存储的策略是：SimpleStrategy，复本因子是replication\_factor，存3个副本。

测试结果：每行数据是3组求平均。

插入的表的结构是：

simplex.songs (id uuid PRIMARY KEY,

title text,

album text,

artist text,

tags set<text>,

data blob);

插入的数据是：

simplex.songs (id, title, album, artist, tags)

VALUES (uuid.toString()，

La Petite Tonkinoise，

Bye Bye Blackbird，

Joséphine Baker，

{'jazz', '2013'} );

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | benchmark数 | 请求总数/benchmark | 并发数/benchmark | TPS/benchmark |
| 1 | 1 | 50W | 25 | 4367 |
| 2 | 1 | 50W | 50 | 3732 |
| 3 | 1 | 50W | 100 | 1801 |
| 4 | 1 | 10W | 50 | 1564 |
| 5 | 1 | 30W | 50 | 2655 |
| 6 | 1 | 500W | 50 | 5653 |
| 7 | 1 | 20W | 20 | 3726 |
| 8 | 1 | 100W | 100 | 2602 |

总结：

1.观察 1,2,3，当请求总数一定时，随着并发数的增加，TPS在减少。

2.观察 2,4,5,6，当并发数一定时，随着请求总数的增加，TPS 在增加。

3. 观察2,7,8，当每条线程写的请求数目一定时，随着并发数增加，TPS先升后降。